パソコンプログラマのためのクリエイティブマガジン
平成3年4月2日第三種郵便物認可　1992年2月1日発行（毎月1回18日）第2巻第2号通巻14号

月刊プログラマーズ・ページ

Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

1992 2 FEBRUARY

定価980円



特集
通信ソフトの作り方
通信用ライブラリ・マクロ言語・ターミナルソフトを作る

付録:5"2HDフロッピーディスク
本誌掲載プログラム（特集「通信用ライブラリ・マクロ言語・ターミナルソフト」ほか）／綴じ込み付録テキストファイル（ATコマンド一覧）／クイズ・ランダムスクリーン解読ツール／シューティングゲーム「SuperDepth」

SE
Shoeisha Co., Ltd.

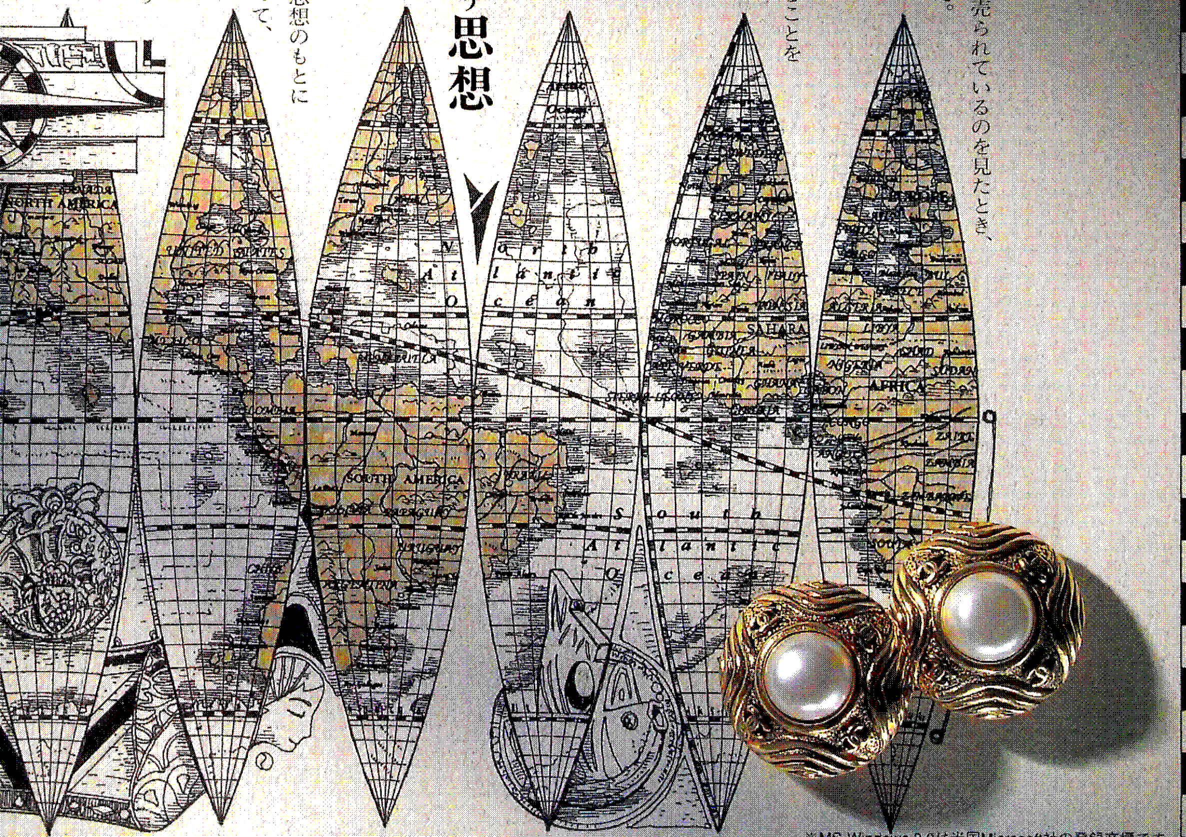
世界標準物語　第二話

シャネルの幸福

パリの路地裏で自分の服のコピーがわずか100フランで売られているのを見たとき、
ココ・シャネルは怒るどころか、勝ち誇ったように言った。
「ほらね、私の人生は成功だったのよ。
誰もが、世界でいちばんエレガントで着やすい
私の服を欲しがっているわ」
またある時、シャネルの服のコピーが世界中に溢れていることを
どう思うかという記者の質問に対し、
彼女はこう答えた。
「多くの女性が私の服を着てくれるってことは、
それだけ種々雑多な服が生まれるってことよ」
一生涯、実業家である前に服飾デザイナーである姿勢を
貫き通したシャネルにとって、
それは真実の思いであったに違いない。

オープンアーキテクチャという思想

IBM PCが、その誕生からわずか数年で
事実上の世界標準機になったいきさつを考えるとき、
ふと、そんなシャネルの逸話を思いおこす。
IBM PCは、オープンアーキテクチャという開かれた思想のもとに
生まれたため、世界中のハード／ソフト・メーカーは競って、
互換機や周辺機器、ソフトの開発にのりだした。
その結果、安価で豊富な互換機が市場に溢れ、
現在6万本以上といわれるソフト資産が蓄積されるという
幸運な事態となった。

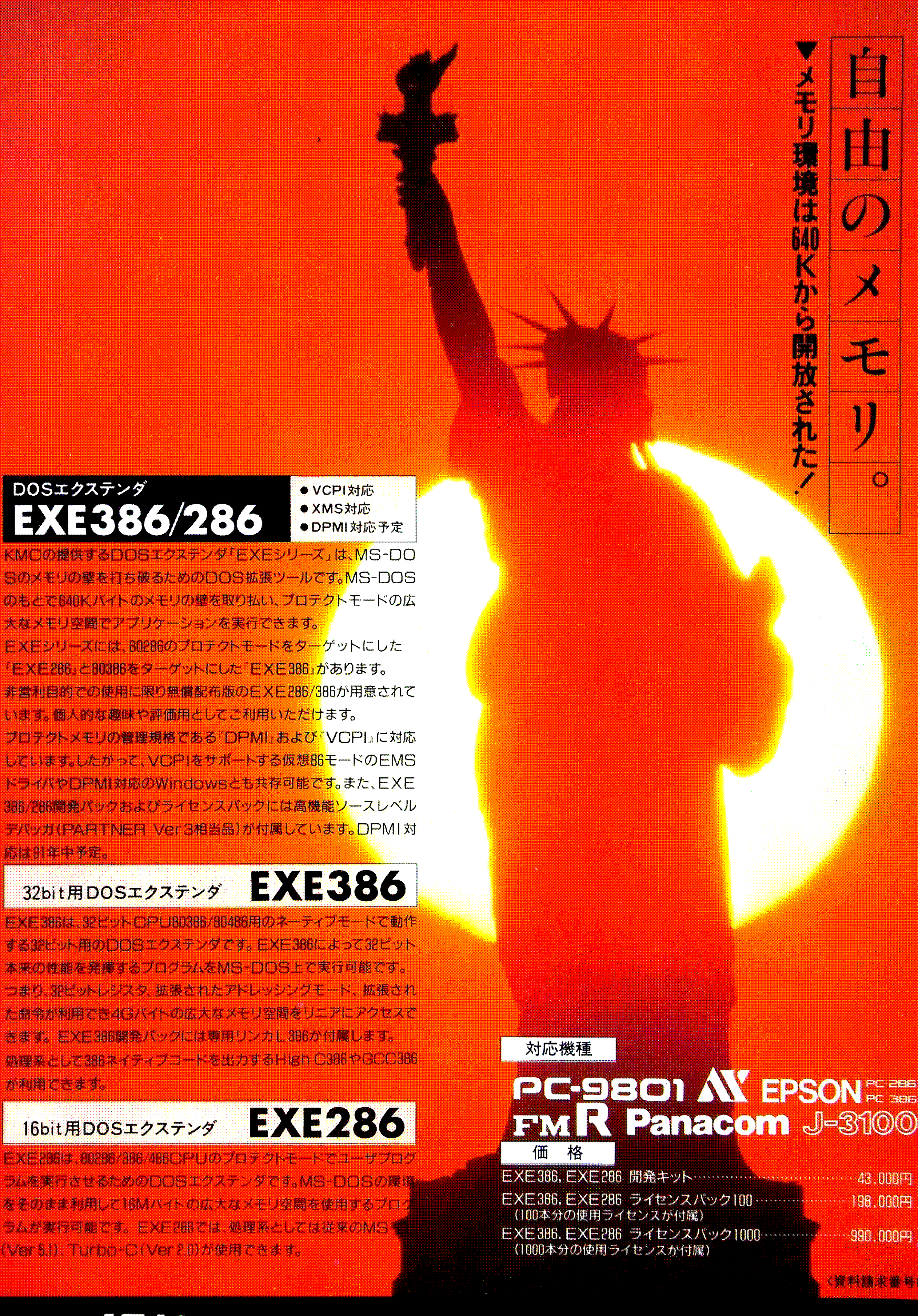


※MS-Windows 3.0は米国Microsoft社の登録商標です。
※IBM PC、VGAは、米国IBM社の商標です。

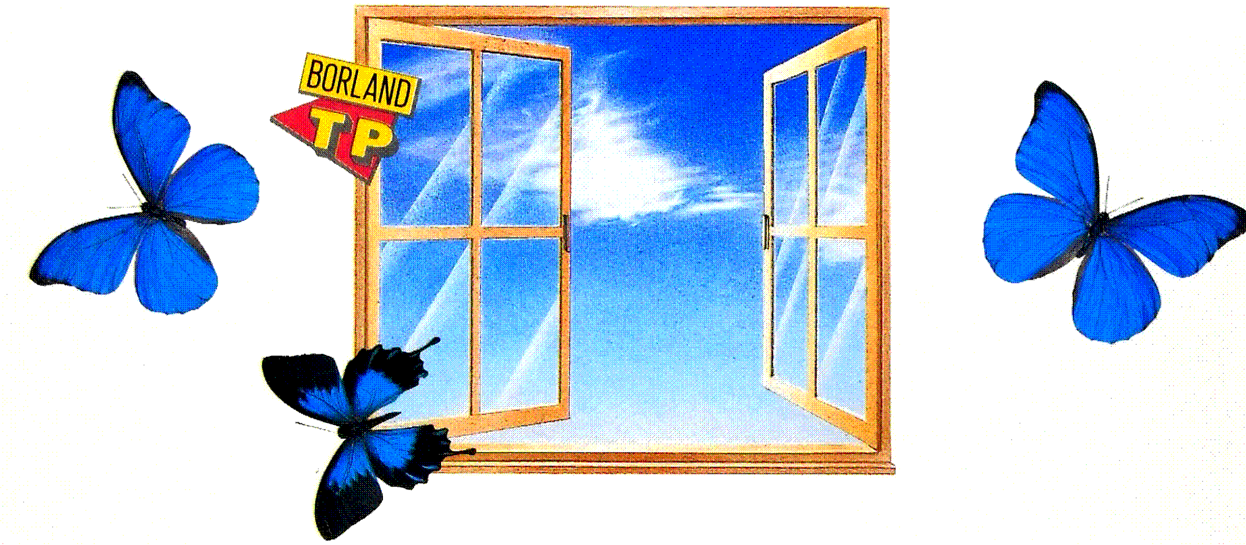
CASIO　HITACHI　KYOCERA　MITSUBISHI
SONY　UNISYS　YHP 横河・ヒューレット・パッカード

社名アルファベット順

お問合わせ：株式会社　翔泳社
TEL. 03-3263-0447
〈資料請求番号P02〉

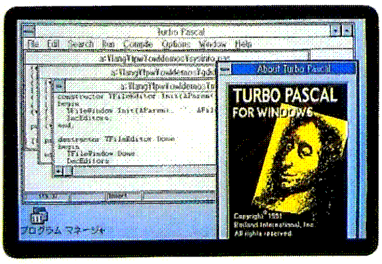

自由のメモリ。
▼メモリ環境は640Kから開放された！
DOSエクステンダ
EXE386/286
●VCPI対応
●XMS対応
●DPMI対応予定
KMCの提供するDOSエクステンダ「EXEシリーズ」は、MS-DOSのメモリの壁を打ち破るためのDOS拡張ツールです。MS-DOSのもとで640Kバイトのメモリの壁を取り払い、プロテクトモードの広大なメモリ空間でアプリケーションを実行できます。
EXEシリーズには、80286のプロテクトモードをターゲットにした『EXE286』と80386をターゲットにした『EXE386』があります。
非営利目的での使用に限り無償配布版のEXE286/386が用意されています。個人的な趣味や評価用としてご利用いただけます。
プロテクトメモリの管理規格である『DPMI』および『VCPI』に対応しています。したがって、VCPIをサポートする仮想86モードのEMSドライバやDPMI対応のWindowsとも共存可能です。また、EXE386/286開発パックおよびライセンスパックには高機能ソースレベルデバッガ(PARTNER Ver3相当品)が付属しています。DPMI対応は91年中予定。
32bit用DOSエクステンダ EXE386
EXE386は、32ビットCPU80386/80486用のネーティブモードで動作する32ビット用のDOSエクステンダです。EXE386によって32ビット本来の性能を発揮するプログラムをMS-DOS上で実行可能です。つまり、32ビットレジスタ、拡張されたアドレッシングモード、拡張された命令が利用でき4Gバイトの広大なメモリ空間をリニアにアクセスできます。EXE386開発パックには専用リンカL386が付属します。処理系として386ネイティブコードを出力するHigh C386やGCC386が利用できます。
16bit用DOSエクステンダ EXE286
EXE286は、80286/386/486CPUのプロテクトモードでユーザプログラムを実行させるためのDOSエクステンダです。MS-DOSの環境をそのまま利用して16Mバイトの広大なメモリ空間を使用するプログラムが実行可能です。EXE286では、処理系としては従来のMS-C(Ver 5.1)、Turbo-C(Ver 2.0)が使用できます。
対応機種
PC-9801 AX EPSON PC-286 PC-386
FMR Panacom J-3100
価格
EXE386、EXE286 開発キット……43,000円
EXE386、EXE286 ライセンスパック100……198,000円
(100本分の使用ライセンスが付属)
EXE386、EXE286 ライセンスパック1000……990,000円
(1000本分の使用ライセンスが付属)
〈資料請求番号

“Easy & True” for Windows.

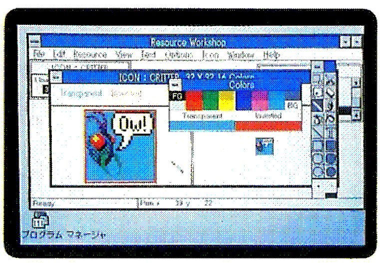


基本は、ターボパスカル——。
完ぺきなWindowsアプリケーション開発環境を提供。

その美しい言語仕様から“コンパイラの貴婦人”と呼ばれ、多くのプログラマーを魅了してきたパスカルが今、「Turbo Pascal for Windows」として生まれ変わりました。デビュー当時、その驚異的な低価格とコンパイルスピードの速さ、デバッガやエディタを備えた世界初の統合環境で人々を圧倒し、瞬く間に標準の地位を確立して以来、ターボパスカルは常に先進の機能を搭載し、言語の世界をリードし続けています。アメリカでは、大学などの教育機関や研究施設でターボパスカルを使用することが常識ですし、ヨーロッパや日本でもビジネスの分野で幅広く利用されています。プログラミングの学習もターボパスカルから始めれば、ベーシックで貧弱な言語で学習したために時間を無駄にする、ということはありません。しかもそのターボパスカルが、複雑で難解だったWindowsのプログラミングを簡単にしてくれるのです。



Turbo Pascal for Windowsには、Object Windowsというアプリケーションの骨格になるクラスライブラリが付属しています。ここからウィンドウやメニュー、ダイアログ、コントロールなどを継承すれば、最初からコードを記述する必要はないのです。オブジェクト指向の拡張がされていないC言語でのプログラムより、格段に素早く、短いコードでアプリケーションを作成できるようになります。Turbo Pascal for Windowsは、自分自身がWindowsのアプリケーションであり、Windowsプログラムのみの作成用に設計されてい



ます。ソースコードの編集、コンパイル、実行、デバッグなど、使いやすさで定評のある統合環境を用い、全てWindows上で行えます。ダイアログ、メニュー、ビットマップ、カーソル、アイコン、文字列リソース、アクセレータ、フォントなどのWindowsリソースを視覚的に作成したり、編集することができるリソースワークショップも付属していますから、高額なSDKを別途購入する必要もありません。Turbo Pascal for Windowsには、Windowsアプリケーション作成に必要なツールの全てが、ひとつのパッケージに含まれているのです。

■動作環境：PC-9800シリーズ(ハイレゾモード対応、LTは除く)、PC-286/386シリーズ/MS-DOS：Ver 3.1以上/MS-Windows 3.0以上必須/マウス：バスマウス/要ハードディスク
■価格………………… 29,800円(税別)



TURBO PASCAL®
FOR WINDOWS



ユーザ向け情報誌
LANGUAGE EXPRESSプレゼント!!

■この広告をご覧になって、製品に関する資料を請求していただいた方全員に、ボーランド発行の情報誌「LANGUAGE EXPRESS」をさし上げます。

●製品に関するお問い合せは、総販売元(株)マイクロソフトウェアアソシエイツ TEL.03-3486-1411までお願いします。

ボーランド株式会社
〒107 東京都港区南青山7-8-1 小田急南青山ビル TEL.03-3486-1400(代)
※社名変更いたしました。

〈資料請求番号002〉

Brief

Full Screen Editor 日本語版

全米のプログラマが認めた実力!
アメリカでは「ブリーフ」が
プログラミング用エディターとして
最も多く利用されています。

C言語で組めるマクロ!

v.3.0 Version UP! いよいよ登場。
プログラミングエディター新世紀。



よりパワフルで、よりエレガントな プログラミングのために……。

●C言語で組めるマクロ 新機能

使い慣れているC言語でマクロが組めますから、習得が簡単でカスタマイズも思いのままです。

●マクロのスクリーン・デバッガ 新機能

ソースコードでデバックできます。もちろんブレークポイントやウォッチ・ドッグもセットできます。

●テンプレート機能〈構文の自動生成〉 新機能

C、Ada、Basic、COBOL、FORTRAN、MODULA2、Pascalで構文を自動生成します。

●300回ものアンドゥ

各ファイルごとに300までの操作を元にもどすことができますので、やり直しの面倒を気にせず、安心して編集できます。

●安全編集

Briefは、一定時間キーを操作しないこと最新の編集内容をファイルに自動保存します。だから編集中に電源が切れても安心です。

●その他の主な新機能

キー操作のファイルへのセーブとリプレイ、処理可能行数を拡張、リジューム機能、Ctrlキー・アサイン(オプション)、メニュー(オプション)etc.

アメリカでも日本でも高い評価

●プログラミング向きエディターを探している人は、もう探す必要はない。(Jerry Pournelle BYTE)
●ソフト開発には強力なツール、300回のアンドゥは大きな特徴。(日経バイト)
●既存のエディターに不満を覚えたら、これしかない。(Oh! FM)
●まるで夢のようなアンドゥだ――おもわず間違うのを期待してしまいそうだ。(The C Users Journal)

■主な機能

◆55ウィンドウ◆無限のファイル◆自動コンパイル◆自動エラー・ジャンプ◆自動インデント◆自動テンプレート◆バッファ書出 切換 削除 リスト表示◆ファイル名の指定 表示 変更◆バックアップ保存◆特殊文字入力◆2行を1行に連結◆ファイル単位の挿入◆増分検索◆各種マーク 始・終点の入換◆ブロックのカット コピー ペースト 印刷 保存 左右へのシフト◆対応する括弧へ移動◆C言語のコメント化 非コメント化◆DOSコマンドの実行◆キー操作記憶 実行◆入力ヒストリー◆英文テキスト整形◆タブのスペース挿入◆拡張子毎に異なった設定ができるタブ◆検索置換パターンの画面からの入力◆英字の大小文字変換 その他

定価 48,000円
(消費税は含まず)

■5機種対応パッケージ

(東芝、NEC、AX協議会、IBM、富士通)
●5 2DD、3.5 2DDデュアルメディア●MS-DOS 3.1以降に対応●英語DOSでは使用できませんが、英語版BRIEFもセット販売いたしております。●お急ぎの方、上記以外のメディア希望の方、英語BRIEFのセット販売希望の方はユーザー・サポート係までご連絡下さい。

日本システム株式会社

●お問合せは 東京都立川市曙町1-18-2一清ビル立川事業所Briefユーザー・サポート係
TEL0425-23-4469 FAX0425-27-6127

〈資料請求番号003〉

PC-PAGE どれをとっても栄養満点！おいしさいっぱいのパソコンブック

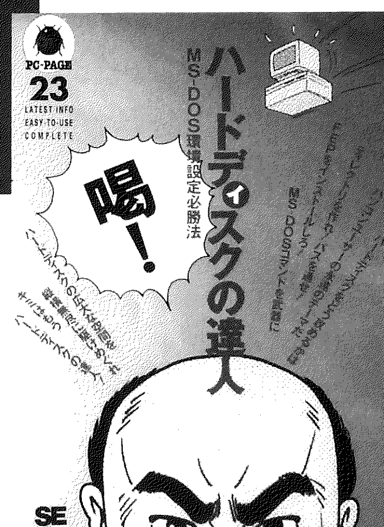


22 パソコンのお医者さん

●A5判 240頁・定価1,500円・SE編集部編

フロッピーディスクが突然読めなくなってしまった！ソフトが起動できない、ハードディスクがクラッシュした………。こうしたパソコンの「病気」に対し、内科、外科、小児科、精神科などの名医たちがトラブルを解決すべく治療のメスをふるいます。まさにパソコン版「家庭の医学」。

PC-PAGE シリーズ最新刊 好評発売中！



23 ハードディスクの達人

●A5判 240頁・定価1,500円・SE編集部編

ハードディスクをフォーマットするところからはじめて、ひととおりの環境を作り上げるまでの過程をおっていきながら、そのつど必要になってくるMS-DOSの知識について丁寧に解説する、いわば自分自身の手でハードディスクをモノにするためのマニュアル。

1	パソコン版標準秋葉原ガイドブック	定価1200円
2	一太郎Ver.4全ファイル解説 （品切れ）	定価1200円
3	林檎百科	定価1200円
4	僕らのパソコン10年史	定価1200円
5	私のパソコンシステム	定価1200円
6	パソコンブックガイド1990	定価1200円
7	BBSの歩き方 （品切れ）	定価1200円
8	東京パソコンマップ1990	定価1200円
9	MS-DOS3.3全ファイル解説	定価1200円
10	パソコンを思想する	定価1200円
11	MS-DOSバッチファイル	定価1500円
12	プロフェッショナルのパソコン利用術	定価1200円
13	PDS白書	定価1500円
14	パソコンとMS-DOS	定価1200円
15	Cの話	定価1500円
16	パソコン見栄講座	定価1500円
17	WINDOWSがやってきた	定価1200円
18	Macintoshグラフィック工房	定価1500円
19	日本語入力読本	定価1500円
20	Windows3.0特別集中講義	定価1500円
21	フリーソフトウェア派宣言	定価1500円

SE Shoeisha Co., Ltd. 株式会社翔泳社 〒102東京都千代田区平河町2-4-14平河町KSビル TEL.03-3263-0447(代表)
翔泳社の本は全国どの書店でもお求めになれます。店頭にない場合は「至急」とお申し出の上ご注文いただければ10日位でお届けできます。

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

特集

通信ソフトの作り方

通信用ライブラリ・マクロ言語・ターミナルソフトを作る

表紙イラスト：日暮修一
表紙 AD：佐々木幹夫(SE)

広告索引	資料請求番号
AXメーカー連合広告	P02
京都マイクロコンピュータ(株)	001
ボーランド(株)	002
日本システム(株)	003
T・ZONE	004
ビレッジセンター	
丸善(株)	P03
ボーランド(株)	P04

人材募集広告

(株)エルテックス

(掲載順)

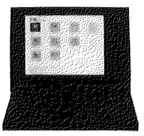
1992 FEBRUARY No.14

IBM Authorized Dealer

Net Ware 取扱開始！

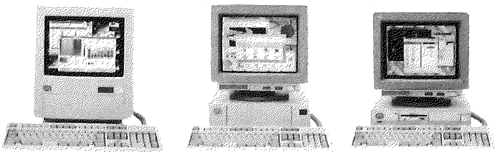
NEWマシンぞくぞく大量入荷!!

PS/55 note



大容量80MBハードディスク内蔵 S18 SV8 S28 SA8
パワフルなDOS/Vノート
80386SX(16MHz)2MB FDD内蔵

PS/55Z 30U　PS/55Z 10S　PS/55Z 10T



XGA採用パーソナルディスクトップ
80386SX(20MHz)2MB or 6MB
パソコンらしくないパソコン？
オシャレ

386SX搭載の
スタンダードPC
DOSV、VGA、ATバス採用
80386SX(20MHz)2MB or 6MB

PS/55Z
最強パフォーマンスPC
XGA高解像度・大容量
80386(20MHz)2MB or 6MB

NEWS とうとう登場！
日本語スーパーVGAドライバー
噂の800×600ドライバがとうとう出た
今すぐT・ZONEへ行こう！

800×600ドットのSuper VGA(S-VGA)を日本語対応にするスクリーンドライバ。
T・ZONEならでは
大特価 ¥2,500

80486/33MHz made in USA
超特価¥498,000　T・ZONE

80486/33MHz
RAM 8MB(max 32MB)
128KBキャッシュ(max 256KB)
3.5" 1.44MB×1/5" 1.2MB×1
200MB IDE HDD
スーパーVGA ET-4000 W/1MB
2シリアル、1パラレル、1ゲーム
拡張スロット×8
ドライブベイ×6
ミニタワー、2ボタンマウス
106キーボード

80386/33MHz ノート made in TAIWAN
限定20台 **超特価¥438,000**　最強ノート

80386/33MHz
RAM 2MB(max 16MB)
32KBキャッシュ
3.5" 1.44MB×1
40MB HDD
32階調VGAバックライト液晶
外部VGA出力
1シリアル、1パラレル
外部テンキー付属
外部101キーボード、外部FDDポート
ニッカドバッテリー(充電2時間、2時間使用)
280W×220D×54H 4kg

80386/33MHz セット
超特価¥398,000
RAM 4MB
64KBキャッシュ
3.5"×1、5"×1
80MB HDD
スーパーVGA 1MB SVGAモニター
2シリアル、1パラレル
1ゲーム
マウス
101キー

Microsoft BALLPOINT MOUSE **超特価¥28,000**

VGA信号をビデオに変換するコンバーター **超特価¥98,000**
320×200〜640×480の信号をS-VHS、ビデオに変換して、テレビやビデオに接続できます。

CREATIVELABS
マルチメディア対応のSOUND BLASTERは必需品です！　特価提供中
SOUND BLASTER

これ1つでマルチメディアが楽しめます。
内蔵CD-ROM
SOUND BLASTER PRO
MIDI KIT
マルチメディアイクステンション他ソフト付
UPGRADE KIT ¥164,800

ステレオ22FM音源
CD-ROMインターフェイス他
SOUND BLASTER互換
Multimedia Sound Standard
PRO ¥57,800

11FM音源
MIDIジョイスティックパワーアンプマイク入力 各種ソフト付
V2.0 ¥34,800

PS/55等のマイクロチャンネル版です。
MCA ¥64,800

MINI-SCSI **超特価¥29,800**　100MB HDDセット **超特価¥138,000**
プリンタポートにつなぐだけでSCSIデバイスが使えるようになります。

Mitac 3060D
106キーセット **超特価¥270,000**
101キーセット **超特価¥258,000**
共通仕様
80386SX(16MHz、3.5" 1.44MB×1、HD 40MB、スーパーVGA(512KB)、スーパーVGAモニター

GAME 時代はゲーム!? TERADRIVEや新作のソフトが山盛りでご来店をお待ちいたしております。
WALT DISNEYの永遠のアイドル、ミッキーマウスがあなたにしゃべりかけます。

MICKEY'S ABC	¥10,800
MICKEY'S 123	¥10,800
MICKEY'S ABC & SOUND SOURCE	¥13,800
SOUND SOURCE(驚異のローコスト音源スピーカーです)	¥6,800
JOYSTICK	¥6,800
ゲームカード(PC/AT)	¥9,800
ゲームカード(MCA)	¥15,000

NOTE ノートといえば、PS/55 note 大特価提供中！

IBM	PS/5523SJ4	超特価¥198,000
SANYO	MBC-18NVH4	~~¥355,000~~
NBCC	386NOTE	¥298,000
NBCC	386LAPTOP	¥266,000
NBCC	286NOTE	¥244,000
NBCC	286LAPTOP	¥220,000

PS/55 note



メモリー
NOTE 2MB ¥21,000
NOTE 4MB ¥40,000

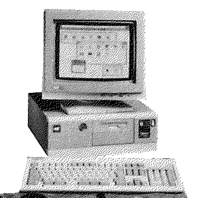
DOS/Vノートおすすめセット　限定

- CPU：80386SX/20MHz
- RAM：3MB
- FDD：3.5"×1
- HDD：40MB
- 外部VGA出力
- A4ファイルサイズ

＋ DOS J5.0/V ＋ シリアルマウス ＋ 一太郎dash

何と！何と！の **超特価¥298,000**

AST RESEARCH JAPAN K.K.
ASTから
DOS/Vマシン
新登場



80386SX/20MHz
FDDモデル **¥308,000**
40MBモデル **¥355,000**
PCvision

VGA DOS/Vの必須アイテムはVGAアダプターです。
豊富な在庫から選べます。

BASIC VGA 640×480 ¥19,800	SUPER-VGA 1024×768 256色 ¥38,000
SUPER-VGA 1024×768 16色 ¥38,000	Dyanbook用VGAアダプタ ¥49,800

VGAモニター DOS/V対応やWINDOWS用として高解像大画面のモニターがほしいあなた、何を選びますか？

NANAO	T660i (ケーブル別)	20" 0.31mm 30〜78kHz	1284×1024	~~¥598,000~~
三菱	XC-98V3	14" 0.28mm 98/VGA	640×460	¥78,000
NEC	MULTISYNC Ⅱ	14" 0.31mm 15.5-35kHz	800×560	¥95,000
IBM	8515	14" 0.28mm VGA、8514/A、XGA		~~¥169,000~~

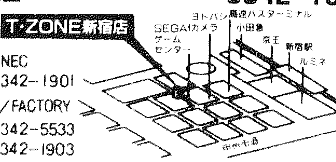
---

the Manhole (日本語版)
それはマンホールからはじまった……
ゲームを超えた
ファンタジーアドベンチャー発売中！



DOS/V対応
IBM PS/55
テラドライブ対応
DOS/V
¥9,800

T・ZONE 新宿店 ☎03 3342-1901

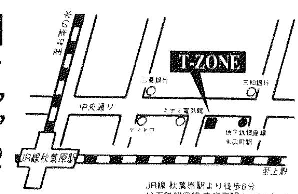


1F APPLE & NEC
03-3342-1901
2F IBM PS/FACTORY
03-3342-5533
FAX 03-3342-1903
〒160 東京都新宿区西新宿1-12-18

T・ZONE PS PLAZA

IBMゾーン
3F ハードウェア
4F ソフトウェア
☎03-3257-2652
FAX 03-3257-2661

営業時間
AM10:30〜PM7:00



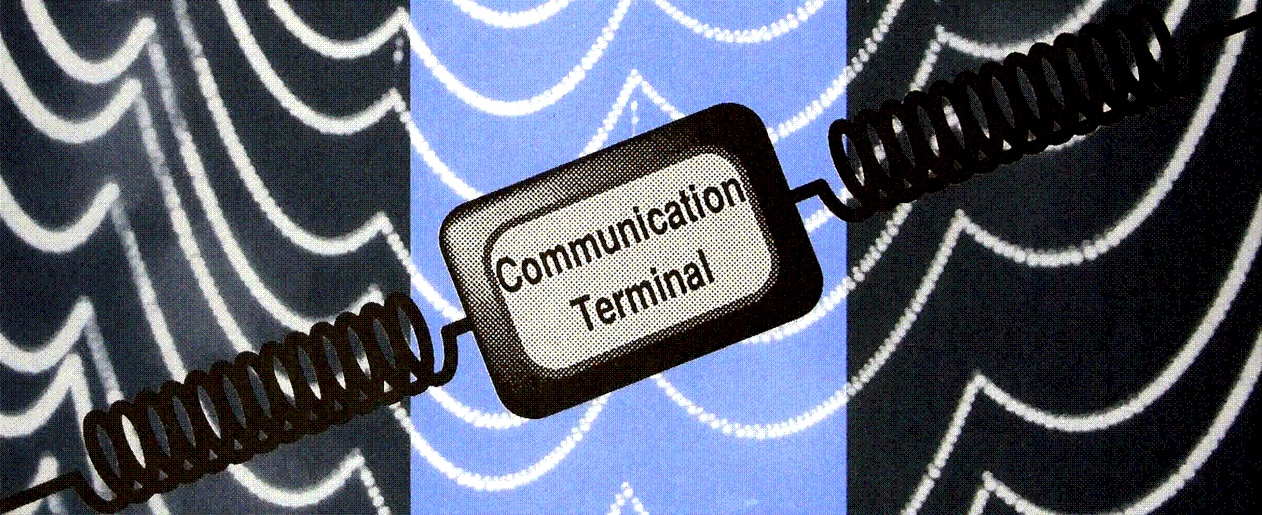
〒101 東京都千代田区外神田4-4-1

T・ZONEスタッフ募集 IBM、Apple、無線のスタッフを募集中です。 アルバイトも募集中！ ☎03-3257-2690

☆T・ZONEの7Fは専門書籍(国内・外)・データーブックの専門フロアです。大量在庫中！ ほしい本は何でも揃います。

特集・通信ソフトの作り方

通信用ライブラリ
マクロ言語
ターミナルソフトを作る



パーソナルコンピュータ上で行われる「通信」には，いわゆる「パソコン通信」からコンピュータ同士を直接接続してのデータ交換や本格的なLANまで，さまざまなものが存在している．本特集ではパソコンに標準として備え付けられているRS-232Cインターフェースを用いた通信に的を絞り，RS-232Cインターフェースの概要からRS-232Cインターフェースを利用するためのライブラリを開発し，さらにこのライブラリを使用して通信用マクロ言語を作成する．最終的には，この通信用マクロ言語を用いてBBSなどにアクセス可能な通信プログラムを完成させる．通信プログラムというとハードウェアがらみのやっかいなものと思いがちだが，本特集のライブラリおよび通信マクロ言語を用いることで，簡単に自分専用の通信プログラムを完成させせることができるだろう．

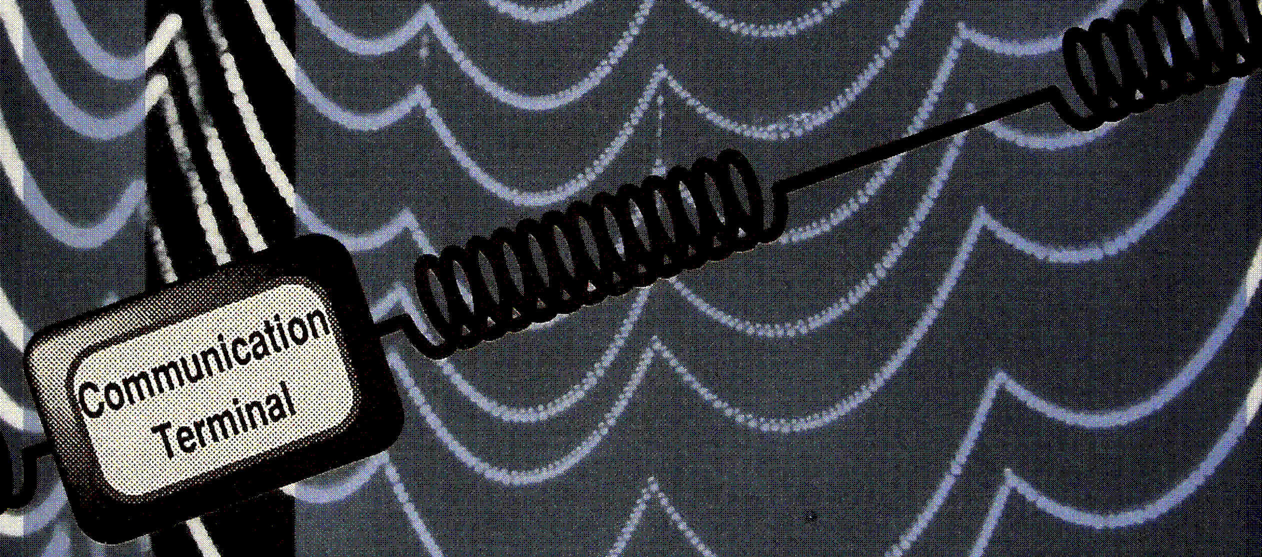
特集・通信ソフトの作り方

第1章 紹介編

通信は難しくない

RS-232C をコントロールするために

株式会社ジャステックラボ　山本博昭



RS-232C は，アメリカの電子工業会（EIA）が決めたシリアル伝送のための規格である．この規格は，電気的仕様から各信号線の種別とその機能，コネクタの仕様などにわたっており，現状のパソコン業界では事実上の標準シリアルインターフェース規格となっている．このため，RS-232C コネクタに接続するモデムなどの機器は大半の機種で共通に接続できる．

通信プログラムの作り方をはじめるにあたって，まずこの RS-232C インターフェースの使用法，通信の原理，通信ソフトに必要とされる機能について解説を行う．

『通信ソフトの作り方』というタイトルで，しばらくお付き合い願いたい．ここを読んでいる方で，実際に「通信」を経験したことのある方はどのくらいの割り合いなのだろうか？　いわゆる「パソコン通信」からコンピュータ同士を直接接続して行うデータ交換や本格的なLANまで，「通信」の定義はほんとうに広範囲におよんでいるため，かなりの方がなんらかの形でコンピュータでの「通信」を利用したことがあるような気がする．かくいう筆者も，数年前から某大手商用ネットワークに加入し，いまでは朝に夕にまるで生活の一部として利用させてもらっている．この原稿さえも電子メールで送るのであるから，まさに典型的なハイテク生活といえよう（単に辺地に住んでいるからという声も聞こえるが）．これもひとえに通信機能の恩恵と感謝する毎日である．

多くの方がなんらかの形で利用されたことがあるであろう通信機能ではあるが，では通信機能を利用するためのプログラムはどうしているのだろうか．市販のものを購入して使っているか，ネットワークに登録されているソフトウェアを入手したという方が大半なのではないかと思う．通信ソフトが自作という方がはたしてどのくらいいるのか，興味深いところである．なかにはプログラムはいろいろと作っているが，通信関係はまだ……という方もいることだろう．通信ソフトはめんどうなものという話はよく聞こえてくる．筆者の乏しい経験から考えるに，通信ソフトがめんどうな理由は，通信制御用LSIのプログラミングの繁雑な点，タイミングや割り込みなど絡んでくる要素の多い点，テスト環境を作ることがめんどうな点などに起因しているように思われる．ならば，これらの要因を1つずつクリアしていけば，いままより少しは手軽に通信ソフトを自作することができるのではなかろうか．

このような背景から，今回はパソコンの標準的な通信インターフェースともいえるRS-232Cをコントロールするソフトウェアをいかにして書くかというテーマでやっていこうと思う．通信関連の一般的な情報から，具体的なLSIの制御などを通じて，なるべく汎用性のあるプログラムパーツとその応用例を紹介していくことになろう．なお，RS-232Cインターフェースを直接制御するプログラムになるため，機種依存部分が出てくるが，今回は筆者の所有環境の都合でNECのPC-9800シリーズを対象にさせていただくことをご了承願いたい．

イントロダクション

「通信」という言葉は，その経験のある場合とそうでない場合で受けるイメージが大きく違ってくると思う．「通信」と聞くとなにか難しそうで怖いものという先入観をもっているような人が身のまわりにいないだろうか？　実のところ，筆者も生まれてはじめてネットワークというものに接続したときは，緊張で手が震えたものだった．いままで自分のパソコンの中だけで暮らしていたのが，電話回線を通じてほかの機械に繋がってしまうのだから，1つまちがうとなにかたいへんなことが起きてしまうのではないだろうかという不安は恐怖に近いものがあった．ひとつ心配しはじめると，世の中のすべてのことが悪いほうへ悪いほうへと考えられてくる．もしもなにかのまちがいで回線を切ることができなくなったら？　まちがった操作をして繋がっているネットワークのホストコンピュータを壊してしまったりしないか？　もちろん，いまではそんな心配はしなくなった．何年間かネットワークを利用しているうちに，繋いだ先のコンピュータが爆発したり，回線を切ることができなくなって果てしなく電話料金が上がり続けることなど普通はありえないということがわかってきたからだ．

というわけで，筆者のパソコン通信機能は趣味に仕事に今日も大活躍してくれている．ところが，人間とは罪深いもので（筆者にかぎったことではないと信じたい），毎日使っているうちにだんだんと使い勝手や機能面で欲がでてくる．この欲がピークに達したとき「それじゃその機能は自分で作ってしまおう」ということになる．「作ってしまおう」という気になったまではいいが，具体的になにをどうすればいいのか，資料や参考文献は多いものの，それらから自分の必要な情報を見つけだしてプログラムの完成までもっていくのはなかなか骨の折れる作業である．そこで「通信ソフトを作る」ために特化してRS-232C関係の情報をまとめるという企画の登場である．RS-232Cというコネクタから無限に広がっていく通信の世界，どうせ使うなら，ただソフトウェアをもってきて利用するだけではなく，その原理を知り，あわよくば自分で通信ソフトを書いてみようではないか．何か1つ作ってみたらこう思われるようになるかもしれない．

「通信ソフトはそんなにむずかしいものじゃない」

パソコン通信用語集
パソコン通信に関する基本的な用語を集めました．

RS-232Cコネクタから広がる世界

さて，NECのPC-9800シリーズのRS-232Cコネクタから話をはじめていこう．おそらくここを読んでいる方の大半は，このシリーズのユーザーであろうと思う(そうでない方のためにも，なるべく一般的な情報や移植の話もおりまぜて進めていくつもりであるのでご容赦願いたい)．おもちのパソコンの本体背面にあるRS-232Cコネクタはいま何かのケーブルで塞がっているだろうか？　買ってから1度もなにも繋いだことがないという方は，いますぐパソコンショップに足を運んで「モデム」と「通信ソフト」をお買い求めになることをおすすめする．パソコンに「もったいないお化け」が棲みついてしまうかもしれない．……まぁ冗談はともかくとして，RS-232Cコネクタにケーブルを繋げている大半の方の用途はモデムとの接続だと思う．そのほかにも，ほかのパソコンと接続してのデータ交換，簡易LAN，シリアルマウスやイメージスキャナの接続など，このコネクタの利用範囲は多岐にわたっていることだろう．

RS-232Cは，もともとアメリカの電子工業会(EIA)が決めたシリアル伝送のための規格である．現在この規格は，電気的仕様から各信号線の種別とその機能，コネクタの仕様などにわたっており，現状のパソコン業界では事実上の標準シリアルインターフェース規格となっている．ディスプレィやキーボード，バスマウスなどは各社独自でインターフェースを設計している関係上，異機種間ではほとんど互換性がないのに対して，RS-232Cコネクタに接続するモデムなどの機器は大半の機種で共通に接続できる．これもこの規格に準拠しているおかげである．

通信ソフトってどういうの？

今度はソフトウェアの側面からRS-232Cを考えてみる．前述したようにRS-232Cコネクタは，電気的特性やコネクタの仕様などが規格化されており，モデムなどの周辺機器が多くの機種で共通に接続できる．ただし，それはあくまでも電気的な特性や信号の定義などハードウェア規格の側面からであって，それをどう実現するかは各社各様の設計になってしまっている．例をあげれば，このインターフェースを制御するためのLSIが各種あって，そのLSIが異なっている機種間（現にPC-9800シリーズとIBMのPC/ATは異なった制御LSIを使用している）ではハードウェア制御のためのソフトウェアが違ってくる．

また，ソフトウェアとのインターフェースの面でも，各社が提供している「RS-232C用のシステムサービス」にさまざまな仕様があるため，おおかたの通信ソフトは機種に依存する結果となってしまう．さらに，信号線の定義はされているが，それをどのように使用するかの厳密な規定がなく，接続する相手によって制御手順が異なってくる場合がある．つまり，その用途に応じて目的の通信相手を制御するためのソフトウェアが必要になるわけだ．たとえば相手が周辺機器であれば，その機器用のドライバ（シリアルマウスドライバやイメージスキャナドライバなど)を組み込み，データ交換や簡易LANなら専用のソフトウェアを使うことになる．

最近あまり話を聞くことのなくなったBASICの「ターミナルモード」も，パソコン本体とケーブルがあれば，それだけで他機との接続ができるもっとも手軽な通信用ソフトウェアである．この素朴な機能でも，2台のパソコンをケーブルで繋いでおいて，片方のパソコンからキー入力した文字がもう片方のディスプレィに表示されたときは深い感銘を受けたものだった．そのほかにも，メガソフト社の「MAX-LINK」のように独自の手法と専用ケーブルで，通常の通信ソフトでは不可能な転送速度をハードウェアの追加なしで実現しているものや，ケーブルで繋いだ端末からコマンドを入力して接続相手のコンピュータ上で動作しているプログラムをテストできるリモートデバッガなどもある．これらのソフトウェアも広義の通信ソフトといえるだろう．

そして，一般にいうところの「通信ソフト」．このごろでは，パソコンショップの店頭でもさまざまな種類の通信ソフトが並んでおり，その個性を競い合っている．各製品とも高性能化・多機能化・操作の簡便化にしのぎをけずっており，おおいに選択に迷うところである．また，この分野ではたくさんのフリーソフトウェアやシェアウェアも登場しており，目的を絞り込んだ個性的な作品や作者自らのきめ細やかなサポートの作品が人気になっている．

パソコン通信用語集
ATコマンド　米Hayes社が提唱した端末からモデムを操作するためのコマンドのこと．コマンドの先頭にATとつけることからこう呼ばれる．

見方を変えれば，ネットワークに接続するための通信ソフトもモデムのためのドライバといえよう．多機能化とはいっても，それはユーザーインターフェースの面であったり，自動運転などの付加機能の充実であって，基本的には RS-232C インターフェースに接続されているモデムをコントロールしての「データ送信」「データ受信」が機能の根幹であることにはかわりないからだ．

通信の原理

話は RS-232C コネクタにケーブルをセットしてモデムを繋ぎ，通信ソフトを起動すると，そこからほかのパソコンやネットワークへ接続できるところまできた．ところで，わがパソコンの RS-232C コネクタから出ているケーブルは，どうやっていま画面に表示されつつある文字をほかのパソコンやネットワークに送り込んでいるのだろうか？　また，どうやってネットワークからのメッセージをこの画面に表示しているのだろうか？　RS-232C を使ったシリアルデータ伝送の動作原理をごくかいつまんでまとめてみたい．

ここで急に話題が変わるが，いま所有されているパソコンが PC-9800シリーズであれば，RS-232C コネクタの近くに，それよりすこし小さめのプリンタコネクタがあると思う．そこにケーブルを繋いでプリンタに接続してワープロソフトや印刷ユーティリティを起動すると，プリンタからデータが印字される．ケーブルを繋いで周辺機器を動かすのだから，これは一見 RS-232C と同じ原理の動作のように見えるが，じつはこの2つには大きな違いがある．結論から先にいってしまうと，

RS-232C は　シリアル通信
プリンタは　　パラレル通信

ということである．シリアル通信というのは，データを1ビットずつ順に送受する方法である．これに対しパラレル通信では，あるデータ単位（8ビットなど）を1度に送受する．この違いを簡単な図にしてみる（図1）．

どうみても，パラレル通信のほうが高速にデータを送受できそうである．シリアル通信の命運やいかにといったところだ．ただしパラレル通信にも弱点はある．シリアル通信の場合，1ビットを送受することに集中してこれを確実に行えばいいが，パラレル通信は同じことを送受データ単位の分だけ行わなくてはならない．このためシリアル通信と同じ品質で通信を行おうとすると，必然的にコストが高くなってしまう．逆にいえば，RS-232C は速度的

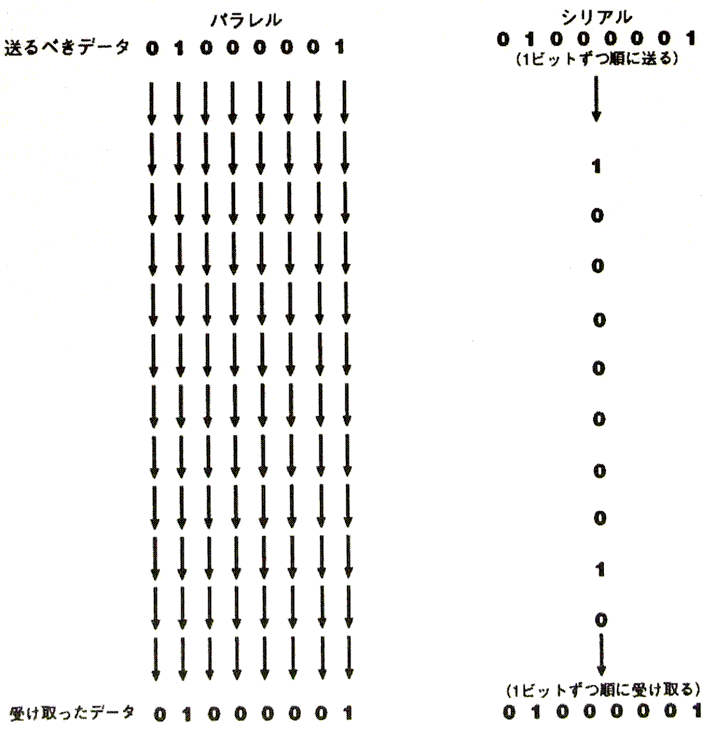


図1
シリアルとパラレルの違い

には劣るものの，低コストで通信のできる手段ということになろう．

1本の信号線を使って1バイトのデータを送るには，図2のように1ビットずつのオン／オフをタイミングを決めて順に伝送していく．当然のことながら，送受信する双方でこのタイミングを合わせておかないと通信はなりたたない．タイミングを細かくしていけば同じ時間により多くのビットを送ることができる．転送速度を表現するためには，1秒間に何ビットのデータを送受できるかというBit Per Second（BPS）が通信速度の単位として一般的に用いられている．

送信するほうは，1バイトのデータをタイミングにしたがって1ビットずつ信号線に送り出し，受信するほうは同じタイミングで1ビットずつ受け取っていって，8ビットになった時点で1バイトデータとして認識する．極端にいえば，線が1本あればシリアル通信ができてしまうのだ．ただし受信と送信を同時に行おうとすると，双方の受信用・送信用の2本のデータ線が必要になる（だから電話線は2線なのだと書こうとしたら，後ろから「嘘を書くな」という声が聞こえた．電話の動作原理はシリアル通信とはまったく違っているそうだ．筆者はそのあと電話の原理について講釈を受けたが，ここでは割愛することにする）．ところがRS-232Cのケーブルを分解してみると線は2本ではなく，かなりたくさんある．データの送受に使う2本の線以外は，接続されている双方の送受信可能状態などをあらわすための補助的な信号線である．通信ソフトを作成する場合は，これらの信号線の制御も考えていかなくてはならない．

送信側は1バイトのデータを1ビットずつ分けて信号線に送り，受信側は1ビットずつ取り込んで8ビットたまったら1バイト分ということだが，こんなことをソフトウェアが自前でやろうとするととてもめんどうであり，このことにかかりきりになってデータを送受するだけで精一杯という事態になってしまう．そのへんはパソコン側も考えたもので，めんどうな1ビットずつの送り出しや取り込みをハードウェアで代行してくれるLSIを搭載している．ソフトウェアでやることは，そのLSIを制御しながら送信のときは1バイトのデータを引き渡し，受信のときは1バイトのデータを引き取ることになる．つまり，パラレル通信と変わらない手間でシリアル通信ができるのである．

ただ世の中はそううまい話ばかりではない．このLSIというのがあんがいくせもので，こんどはそのLSIのプログラムがなかなか厄介である．しかもパソコンの機種によって採用しているLSIが違っていたりして，通信プログラムの機種依存の一因となっている．

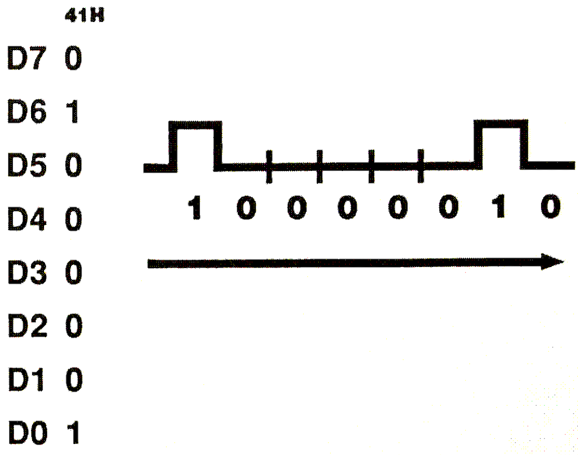


図2
41H（最上位ビットから順に送る）

これだけあれば通信ソフト

では，実際にRS-232Cからデータを送受するプログラムを考えてみることにしよう．広く使われているネットワーク接続用の通信ソフトは実にさまざまな機能をもっており，その構造はたいへん複雑に見えるが，直接通信に関係ある基本部分だけを抜き出すとだいたい図3のようになるだろう．

図3を見ると，RS-232Cの制御に必要なソフトウェアモジュールは次のように分類できる．

- ハードウェアの初期化に関するもの
- モデムの制御に関するもの
- RS-232Cからのデータ受信
- RS-232Cへのデータ送出

パソコン通信用語集
CCITT V.22勧告　300，1200bps全2重モデムの変調方式，ハンドシェイク手順，リトレイン手順など接続の規格．

・RS-232C からの状態変化に対する応答
・内部状態の変化による RS-232C への状態設定
・通信終了時の後処理

このうちモデム制御に関しては，モデムへのコマンドを RS-232C を通じてモデムに送出するわけだから，本体からの送信という意味では RS-232C へのデータ送出となんら変わることはない．それを考慮してさらに機能別に分けてみると

・初期化および終了処理
・データの送信と受信
・RS-232C の状態監視と状態設定

以上，たった３つになってしまった．こういうふうに機能別にブロック化していくと，すぐに思い浮かぶのは，「これをライブラリ化しよう」ということである．ライブラリ化の利点は

・処理をブラックボックス化できるので，アプリケーションの開発や試験，メンテナンスが楽になる．
・機種依存部分を閉じ込めておけば，他機種への移植が楽になる．
・複数のアプリケーションで共用することによって，プログラム作成が楽になる．

楽になる話ばかり書いてしまったが，注意すべき点もある．汎用性をもたせるためにライブラリのパーツはできるだけ単機能にすること，インターフェースをしっかり設計して強度の高いモジュールにすることである．アプリケーションを作ってからライブラリの仕様を変更するのは悲惨である．こういったことを念頭においたうえで，次章では具体的なライブラリ仕様とそのプログラミングについて書いていくことにしよう．

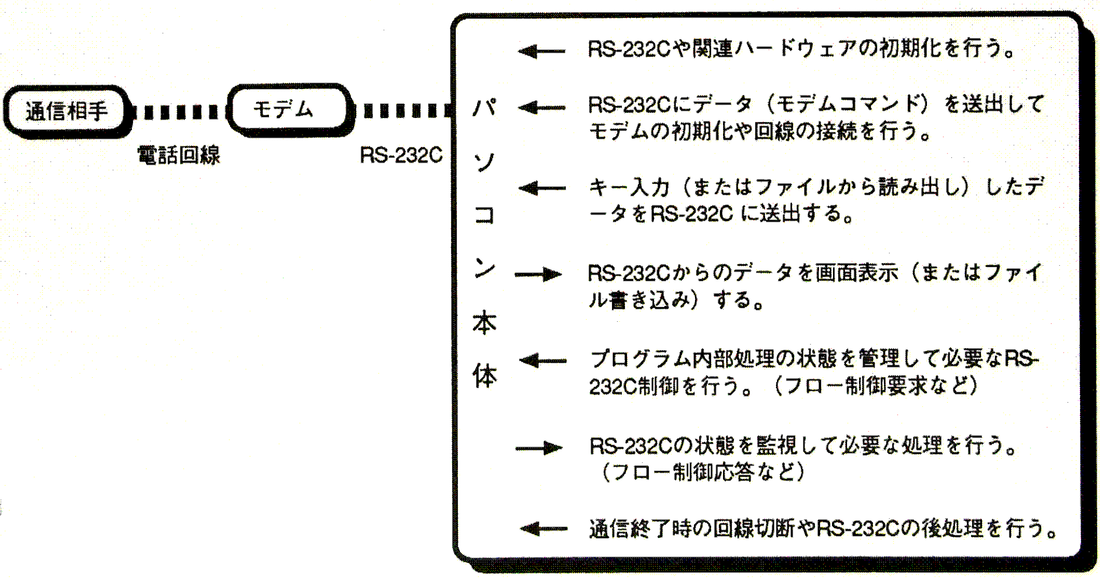


図３
ネットワーク接続用通信ソフトの
RS-232C 関係処理図

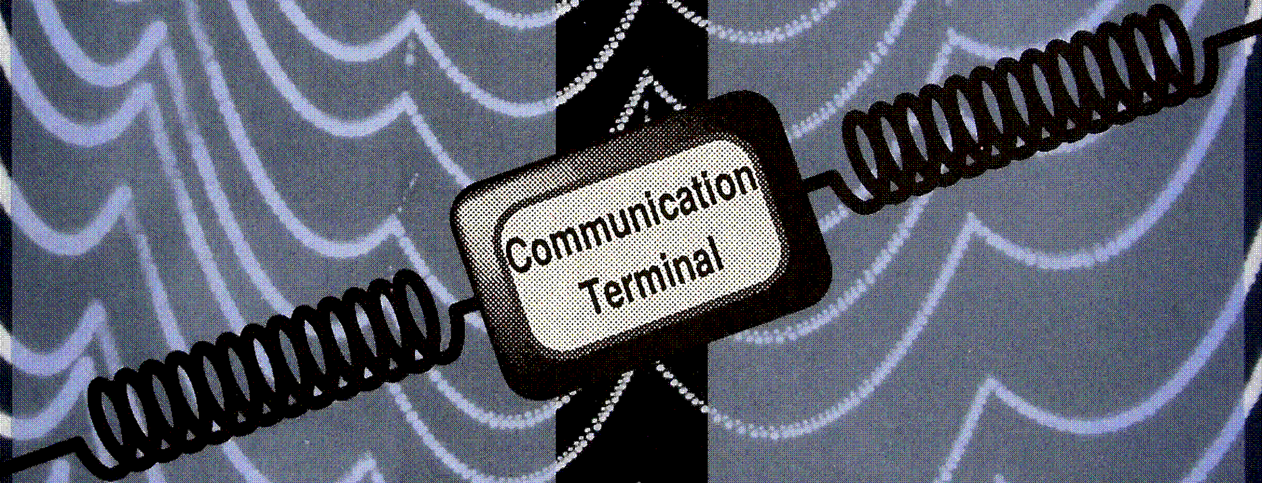
パソコン通信用語集
CCITT V.25bis勧告　モデムをパソコンから操作するためのコマンドの規格．ただし一般的にはHayes社の提唱したATコマンドが普及している．

第2章 基礎編

通信ライブラリを作ろう

LSIや信号線を制御する

株式会社ジャステックラボ　占部貞之・山本博昭



実際に通信プログラムを作ろうとする際に，LSIの制御や信号線の監視などを一から自分で作り上げようとすると，その作業量はかなりのものになる．気楽にとりかかるには，ちょっと荷が重いといえるだろう．そこで，こうした繁雑な制御をライブラリ化し，それらを使用して手軽に通信ソフトを組み上げられるようにする．
また，ライブラリを作成するのに必要なハードウェアの情報およびライブラリ関数の処理の内容についても解説を行う．

NECのPC-9800シリーズで通信をやろうということで，RS-232Cでの通信機能のアウトラインを紹介してきたが，いざ実際にプログラムを作ろうとして，LSIの制御や信号線の監視などを1から自分でやってみようと思ったら「うーん……」ということになってしまいそうである．この章では，こういったこまごまとした制御をライブラリ関数としてパーツ化し，それを組み合わせて使うことによってより手軽に通信ソフトを組み上げられるような通信ライブラリをまとめることにする．ただ，パーツ化するとはいっても内部でどう動いているのかまったくわからないのではおもしろくないので，ライブラリを作成するのに最低限必要なハードウェアの情報やライブラリ内の各関数の処理概要を解説してからライブラリリストを掲載することにしたい．

ハードウェアを知ろう

PC-9800シリーズで動作する通信ライブラリを作成するわけだが，この機種で通信機能をもったプログラムを作成するためには，RS-232C関係のハードウェアだけではなく，割り込みコントローラやタイマ関係の制御などの処理も必要となってくる．ここでは，そういった関連ハードウェア制御もふくめて，PC-9800シリーズでRS-232Cを使用する場合に必要なハードウェアの概要をまとめる．

今回制御が必要となるLSIは，図1のような構成になる．

以降，それぞれの周辺ごとに制御の実際を見ていくことになるが，これらのLSIはどれも非常に多彩な機能や複雑なモードをもっており，それをすべて詳説してしまうとそれだけで1冊の本が書けてしまうし，それは本章の本題ではないので，ここでは今回の通信ライブラリに必要な部分のみの記載とする．本章で記述した以外のLSI制御方法が知りたいという場合は，ほかの参考文献を参照してほしい．

【RS-232C制御：8251】

PC-9800シリーズが搭載している8251は，「同期／非同期通信用LSI」(Universal Synchronous/Aynchronous Receiver/Transmitter = USART) と呼ばれ，通信制御用LSIの代表選手的存在である．このLSIは

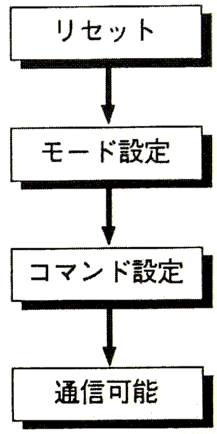


図2

多彩な機能をもっており，8251の説明書などを読むと，全部の機能を網羅しているために，そのプログラミングはたいへん複雑に書いてある．

今回は「98でRS-232Cを使う」という命題にしぼりこんで，通信ライブラリを書くうえで必要になるプログラミングのみをまとめてみる．

8251のプログラミングの大きな流れは図2のようになる．

プログラミングは，8251へのポートに出力することによって行う．PC-9801の場合，ポートは表1のようになっている．

では，順に具体的なプログラミングを解説する．

概要	LSI名称	用途
RS-232C制御	8251	：RS232Cによるシリアル通信の制御
割り込みコントローラ	8259	：8251からの送受信割り込みの制御
システムポート	8255	：送受信の割り込み発生の許可/不許可制御
ボーレート制御	8253	：通信ボーレートの制御

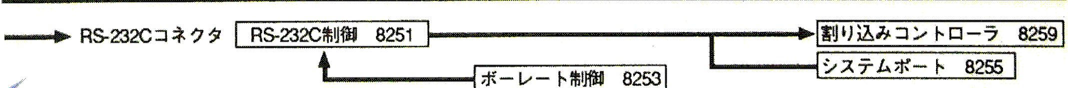


図1

リセット

8251がその時点でどういう状態になっているかを問わず，モード設定待ちになるように内部リセットを送出する．図3のように，内部リセットを行うためには，コマンド設定でD6ビットをONにして出力すればよい．ここで1つ問題がある．前述したように，8251はモード設定を行うポートとコマンド設定を行うポートが同一であり，内部的に「内部リセット直後の出力のみをモード設定へのものとし，それ以外はコマンドである」というロジックになっている．このため，いきなりコマンド設定を送出しても8251の状態によってはモード設定とみなされる危険性がある．これを回避するためにリスト1のようにして，確実に内部リセットを実行できるようにダミーのコマンドを送出することになる．

注：各ポートへのコーディング例が今後も出てくるが，コーディングサンプルのため入出力の際に必要となる遅延などをはぶいてある．実際に動作するロジックは，ライブラリのリストを参照されたい．

表1

ポート番号	入出力区分	内　容
32H	出力	8251モード／コマンド
32H	入力	8251ステータス
30H	出力	8251データ送信
30H	入力	8251データ受信

リスト1

```
mov     al,0
out     port32_8251,al      ; ダミーの出力を行って確実に
out     port32_8251,al      ; モード設定待ちにする
out     port32_8251,al
mov     al,01000000b
out     port32_8251,al      ; 実際の内部リセット
```

モード設定

内部リセットが行われると，8251はモード設定待ち状態になる．ここで，通信のモードを設定する．ここで必要な情報は次のものである．

- **キャラクタ長**
- **ストップビット長**
- **パリティチェックの有無**
- **パリティの極性**
- **ボーレート**

いずれも通信ソフトを使ったことのある人ならおなじみの条件であろう．ここでは詳しい説明をはぶくが，要は「相手側と同じ条件を設定しなければならない」ということである．

具体的にモードを設定してみよう．現在国内で運用されているパソコン通信のほとんどの通信条件は次のとおりである．

- **キャラクタ長　　8ビット**
- **ストップビット　1ビット**
- **パリティチェック　なし**

また，8251へのモード設定のときの「ボーレート」は，実際のボーレートを設定するわけではなく，8251へ入力されてくるクロックをどのように使うかという指定である．ここでは「x16」を固定で使うものとする．

さて，上記のような通信条件を設定するためには図5にしたがって表2のようにする．つまり，表2のパターンをモードとして設定すればよいことになる（リスト2）．

注：通信条件は「こちら側」と「相手

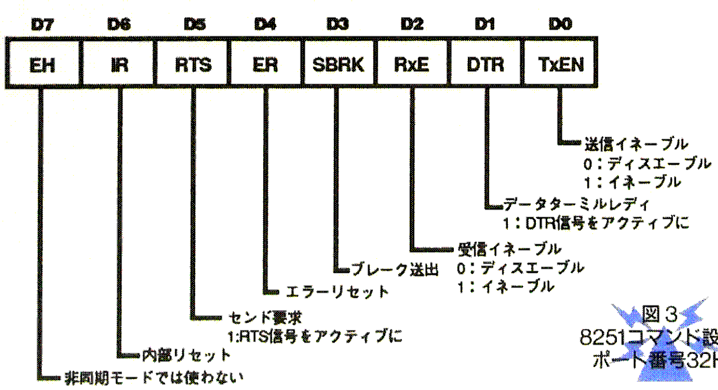


図3
8251コマンド設定
ポート番号32H

パソコン通信用語集
CCITT V.32ter勧告　19200bps全2重モデムの変調方式，ハンドシェイク手順，リトレイン手順など接続の規格．

表2

・キャラクタ長8ビット	L2 = 1 , L1 = 1
・ストップビット1ビット	S2 = 0 , S1 = 1
・パリティチェックなし	PEN=0 (パリティなしなので, EPは意味をもたない)
・ボーレートx16	B2 = 1 , B1 = 0

S2	S1	EP	PEN	L2	L1	B2	B1
0	1	0	0	1	1	1	0

リスト2

```
mov    al,01001110b
out    port32_8251,al      ; モード設定
```

側」の条件が合っていなければいけないと書いたが，実際の話はどうだろうか？　双方のボーレートが合っていない場合は繋がらないのだが，じつは「キャラクタ長」と「パリティ」はその合計数が合っていれば繋がってしまうのである．たとえば，図4のような場合である．

この例だと，ビットの合計数が同じため，一見通信ができているかのように動作する．もちろん条件が一致していないのでまともなデータを送受できるはずはなく，めちゃめちゃな文字がやり取りされるだけなのであるが，条件が合っていないと通信自体ができないと信じ込んでいると，ほかの要因を考えて苦労することになりかねないので注意が必要である．

図4

キャラクタ長7ビット 偶数パリティ ⟷ キャラクタ長8ビット パリティなし

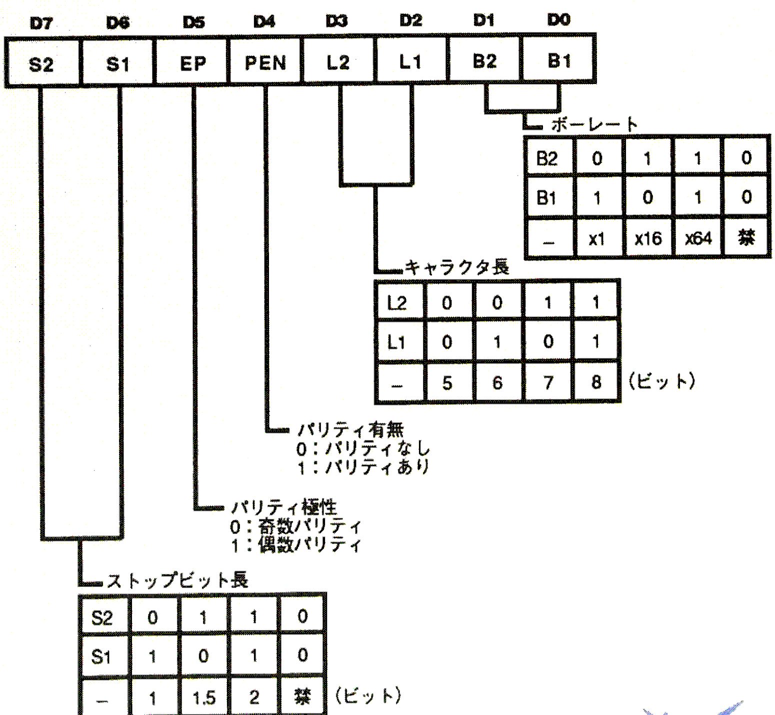


図5
8251モード設定
ポート番号32H

コマンド設定

8251のモードが確定したら，コマンドを設定して通信が可能な状態とする．前述の図3のコマンド設定をいま一度確認してほしい．ここで実際の通信に必要なビットは

- 送信イネーブル
- 受信イネーブル
- センド要求
- データターミナルレディ

の4ビットである．ここでは，これらすべてのビットを有効として通信が送受信ともに可能な状態に設定する（リスト3）．

これで8251への設定は終りである．意外なほど簡単である．もっともさきほどから書いているように，このLSIはまだまだ機能をたくさんもっていて，複雑な制御もできるようになっている．ただ，98でRS-232Cを用いた通信を行う場合にかぎると，このようなシンプルなプログラミングで実用になるということである．

【割り込みコントローラ：8259】

PC-9800シリーズでは，RS-232Cを通じてのデータ送受の際，次のいずれかの条件でハードウェア割り込みを発生させることができる．

- データを受信したとき
- データ送信が可能になったとき
- 8251の送信データバッファが空になったとき

RS-232Cのハードウェア割り込みは，割り込みコントローラ8259を介して，INT 0Chとして発生する．また，割り込みコントローラは図6のようにほかにもさまざまな割り込みが接続されており，そのおのおのの割り込みの発生を禁止・許可するためのマスクレジスタをもっている．マスクレジスタは，目的の割り込みの該当ビットを“0”にすることによって割り込みが可能状態となり，“1”にすることによって不可能状態となる．

当ライブラリの初期化ルーチンでは，RS-232Cのハードウェア割り込みを可能状態にしている．その際，ほかの割り込みのマスクビットに変更を与えないように1度マスクレジスタを読み込んでRS-232Cのビットのみを“0”にしたうえで，そのデータを出力する．

LIST

リスト3

```
mov     al,00100111b
out     port32_8251,al      ; コマンド設定
```

リスト4

```
in      al,port02_8259
and     al,11101111b           ; ＲＳ－２３２Ｃのビットのみを０にする
out     port02_8259,al         ; マスク設定
```

リスト5

```
mov     al,00100000b           ; EOIｺﾏﾝﾄﾞ
out     port00_8259,al
```

図6に従って，RS-232Cの割り込みを可能状態にするコーディングはリスト4のようになる．

また，ハードウェア割り込みの処理が終了したときに割り込み処理の終了通知(End Of Interrupt = EOI)を割り込みコントローラに送出する．EOIは図7によってリスト5のような出力を行う．

【システムポート：8255】

8255は本来汎用のパラレルインターフェース制御用のLSIであるが，98ではこのLSIをブザーのON/OFFやディップスイッチの読み取りなど，さまざまな用途に使用しており，これをシステムポートと呼んでいる．前述したRS-232Cの割り込みもこのシステムポートで，次の割り込み要因について個別に禁止・許可ができる．

- データを受信したとき
 (RxRE)
- データ送信が可能になったとき
 (TxRE)
- 8251の送信データバッファが空になったとき
 (TxEE)

図8にしたがって，送信割り込み(TxRE)を禁止する場合のコーディングは，リスト6のようになる．

図8のポートの操作では，ほかのビットを変化させないために，1度ポートの現状態を入力してそのデータ中の目的のビットを操作したあとで出力しなおす必要がある．このポートデータの目的のビットのみを操作するために図9のような専用のポートが別に設けられている．こちらのポートでは，目

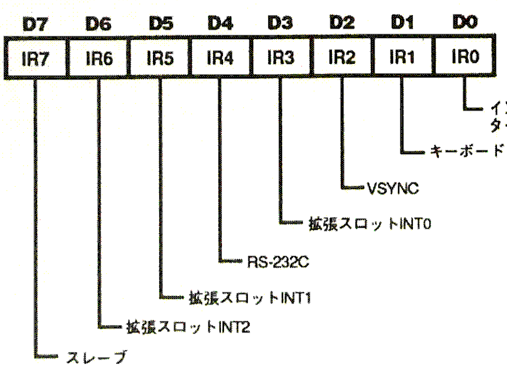


図6
8259マスク設定
ポート番号02H
(0でマスク解除)

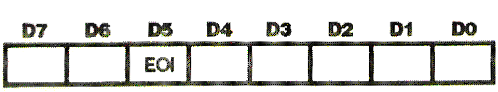


図7
8259オペレーション
コマンドポート番号00H
(関係ビットのみ)

パソコン通信用語集
CCITT V.42bis勧告　データ圧縮手順の規格．CCITT V.42を前提として，その上でデータ圧縮を実現する．

的とするビットのみをON/OFFすることができるため，出力1回のみで処理ができる．

図9のポートを使って，上記と同じ送信割り込み（TxRE）を禁止する場合のコーディングをしてみる（リスト7）．

図10のポートは読み出し専用で，CD，CSという回線状態を示すビットの状態がリードできる．

【ボーレート制御：8253】

PC-9800シリーズは，通信の際にボーレートのタイミングを作り出すためにタイマLSIを使用している．タイマLSI(8253)は，3つのタイマカウンタを内蔵しており，それぞれ表3のように使用されている．

8253もほかの周辺LSIと同様に，さまざまな動作モードをもつ汎用タイマであるが，例によって，通信に必要な機能を設定するということで，今回は次のようなモードで使用することになる（リスト8）．

- **タイマ#2**
- **モード3**
- **バイナリカウント**
- **下位→上位バイトの順でのリードライト**

タイマLSIに設定すべきカウンタ値は，図11のようにして計算する．

具体的な計算をする前に，98では8253へのクロックがCPUの動作クロックによって変化することを念頭におかなければならない．98にもいろいろ

LIST

リスト6

```
in      al,port35_8255
and     al,11111011b
out     port35_8255,al
```

リスト7

```
mov     al,00000100b
out     port37_8255,al
```

リスト8

```
mov     al,10110110b
out     port77_8253,al
```

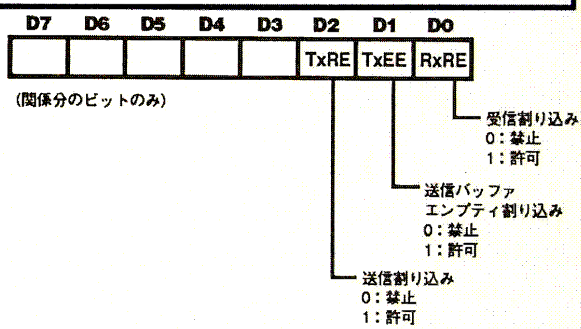


図8 ポートC ポート番号35H

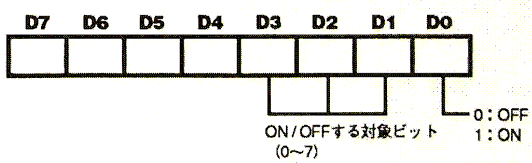


図9 ポートCのビットをON/OFFする専用ポート番号37H

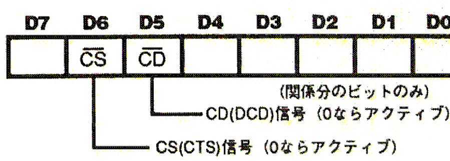


図10 ポートB（読み出し専用） ポート番号33H

カウンタ値=8253へのクロック周波数÷ボーレート÷8251へのボーレート設定

図11

表3

タイマ#0	：	インターバルタイマ（タイマBIOSで使用）
タイマ#1	：	スピーカ周波数設定
タイマ#2	：	RS-232Cボーレート設定

表4

システムエリア　0000:0501のD7ビットが	0のとき	2457600Hz
	1のとき	1996800Hz

な機種があるので，CPUクロックが何MHzのときはいくつということはいちがいにいえないが，表4のようになっている．

前述したように，当ライブラリの場合には8251へのボーレート設定がx16で固定になっているため，2,457,600Hzでボーレート9600の場合は図12のようになる．

こうして計算したカウンタ値を図14のポートに出力するコーディングをしてみる(リスト9)．LSIの設定のときにもあったように，下位バイト→上位バイトの順に1ワード送出する．

98のRS-232Cインターフェース

具体的な信号名も出てきたので，ここでPC-9800シリーズが標準でもっているRS-232Cインターフェースの仕様をまとめておくことにする．RS-232Cインターフェースは，図15のように25ピンのコネクタになっている(そのうち9ピンは未使用になっている)．

ライブラリに必要な機能について

それでは，いよいよ通信ライブラリを具体的に作っていくことにしよう．今回のライブラリは，C言語で作成する通信ソフトから呼ばれることを想定してインターフェースを決めている．

通信ライブラリに必要な機能として

2457600÷9600÷16＝16 ← カウンタ値

図12

リスト9

```
mov     ax,16
out     port75_8253,al
mov     al,ah
out     port75_8253,al
```

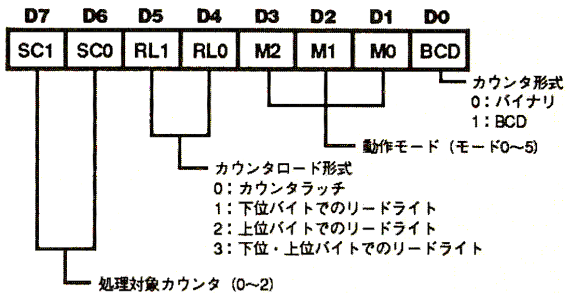


図13
8253モード設定
ポート番号77H

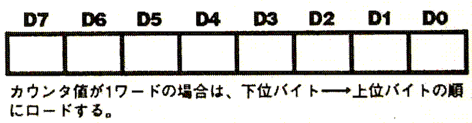
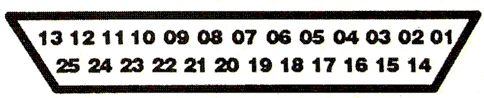


図14
8253カウンタ設定
ポート番号75H



ピン番号	信号名	概要
01	GND	フレームグラウンド
02	TXD	送信データ（Tx Data)
03	RXD	受信データ（Rx Data)
04	RTS	送信要求（Request To Send)
05	CTS	送信可(Clear To Send)
06	DSR	データセットレディ(Data Set Ready)
07	GND	グラウンド
08	DCD	キャリア検出(Data Carrier Detect)
09	×	未使用
10	×	未使用
11	GND	グラウンド
12	×	未使用
13	GND	グラウンド
14	GND	グラウンド
15	TXC(2)	送信クロック(Tx Clock)
16	×	未使用
17	RXC	受信クロック(Rx Clock)
18	×	未使用
19	×	未使用
20	DTR	データターミナルレディ(Data Terminal Ready)
21	×	未使用
22	RI	被呼表示(Ring Indicator)
23	×	未使用
24	TXC(1)	送信クロック(Tx Clock)
25	×	未使用

注：信号名は、旧JIS規格のものを用いた。モデムなどの表示で新JISを使用しているものもあるので以下にその対照をまとめる。

旧JIS	新JIS
TXD →	SD
RXD →	RD
RTS →	RS
CTS →	CS
DSR →	DR
DTR →	ER

図15

パソコン通信用語集
CCITT　国際電信電話諮問委員会（Consultive Committee for International Telegraph and Telepone）の略称で，電信電話やデータ通信などの国際標準化を行っている機関．

は先ほどまとめたように，

- RS-232Cインターフェースの初期化と終了処理
- データ受信／データ送信
- 各種信号線の制御と監視

といったところが基本モジュールとなる．このほかにモデムや通信相手が無反応になったときのためのタイムアウト管理なども必要になってこよう．

表5にライブラリの関数一覧をまとめる．各関数は，汎用性をもたせるためにできるだけ単機能とした．これらの関数を組み合わせて使用することによって，ほとんどのRS-232C制御ができるはずである．

表5

番号　関数名	機能
初期化，終了処理	
01：rs_initialize	RS-232C初期化（通信使用開始）
02：rs_finish	通信使用終了
データの送受，チェック	
03：rs_send_data	送信バッファに1バイト書き込む
04：rs_receive_data	受信バッファから1バイト取り出す
05：rs_peek_data	受信バッファから1バイト読み出す
06：rs_send_buffer_space	送信バッファ余裕バイト数取得
07：rs_receive_count	受信済みデータ数取得
08：rs_send_count	送信待ちデータ数取得
各種信号等の制御，チェック	
09：rs_xcontrol_on	Xコントロールモード ON
10：rs_xcontrol_off	Xコントロールモード OFF
11：rs_rts_on	RTS（RS）　ON
12：rs_rts_off	RTS（RS）　OFF
13：rs_dtr_on	DTR（ER）　ON
14：rs_dtr_off	DTR（ER）　OFF
15：rs_break_on	ブレーク信号　ON
16：rs_break_off	ブレーク信号　OFF
17：rs_xcontrol_check	Xコントロールモード検査
18：rs_dcd_check	DCD（CD）検査
19：rs_cts_check	CTS（CS）検査
20：rs_dsr_check	DSR（DR）検査
タイムアウト監視	
21：rs_set_timer	タイムアウト監視用タイマ設定
22：rs_chk_timer	タイムアウトチェック

通信ライブラリ機能仕様書

前段で一覧にしたライブラリ関数の詳細を解説する．C言語で書かれたアプリケーションプログラムから呼ばれることを前提として，呼び出し方と戻り値を記述する．動作概要では，その関数の処理の流れを簡単にまとめる．関数を使う際には直接関係しない事柄もあるが，通信機能を制御するプログラムを作るうえでの参考になれば幸いである．

表6がアプリケーションプログラムから呼び出すことのできる関数群の仕様である．

表6

01：RS-232C初期化（通信使用開始）

【呼び出し方】 void rs_initialize(void)

【戻り値】

なし

【動作概要】

通信条件に合わせて，RS-232C制御用LSI(8251)および割り込み関係，タイマ関係などの初期化を行う．当ライブラリの各関数を使用する場合は必ず事前にこの初期化を行わなくてはならない．設定する通信条件は次の通り．

ボーレート	(75bps～19200bps)
キャラクタ長	(7または8ビット)
ストップビット長	(1　1.5　2のいずれか)
パリティ種別	(なし，偶数，奇数のいずれか)
Xコントロール	(する，しないのいずれか)

この関数を呼び出す前準備として，まず呼び側で以下のエリアに通信条件を設定する．これらのエリアは本ライブラリのワークエリアとして確保してあるものである．使用するアプリケーションプログラムから外部参照宣言をして設定すること．

パソコン通信用語集
CD信号　モデムから端末へ，キャリアを受信中（＝相手モデムと接続中）であることを知らせる信号（データキャリア検出）．

表6

変数名	意味	セットする値
rs_baud_rate	ボーレート	0 = 75 bps
		1 = 150 bps
		2 = 300 bps
		3 = 600 bps
		4 = 1200 bps
		5 = 2400 bps
		6 = 4800 bps
		7 = 9600 bps
		8 = 19200 bps
rs_data_length	キャラクタ長	2 = 7ビット
		3 = 8ビット
rs_parity_bit	パリティ種別	0 = なし
		1 = 奇数
		3 = 偶数
rs_stop_bit	ストップビット長	1 = 1ビット
		2 = 1.5ビット
		3 = 2ビット
rs_xcontrol	Xコントロール	0 = しない
		1 = する

以下に関数内の初期化手順を記述する．

1．すでに初期化済みであれば，いったん使用を終了させるために，後述のrs_finishを実行する．
2．送受信バッファの初期化を行う．
3．割り込みコントローラおよびシステムポートの送受信許可状態を示すポートの現時点での内容を保存しておく（終了時に復帰するため）．
4．指定されたボーレートに従って，タイマLSIを初期設定する．
5．指定された通信条件にしたがって，RS-232C用LSIを初期設定する．
6．割り込みコントローラのRS-232C割り込みマスクをオープンする．
7．RS-232Cハードウェア割り込み（INT 0Ch）の割り込みベクタを保存し，当ライブラリでの割り込み処理ルーチンを代わりに設定する．
8．システムポートへ受信許可の状態設定を行う（ここでは受信のみ許可とする．送信は，送信処理の中で許可する）．

02：通信使用終了

【呼び出し方】　void rs_finish(void)

【戻り値】

なし

【動作概要】

初期化のときに記述したが，当ライブラリではRS-232Cのハードウェア割り込みを引き込んだり，RS-232C用LSIへの設定などをしているため，通信機能の使用を終了した時点で（大概の場合は，このライブラリを使用するアプリケーションプログラムの終了時に）通信使用の終了処理を行う必要がある．当関数は，この終了処理を行うものである．

ライブラリで使用していた割り込みを元の処理ベクタに戻すとともに，RS-232Cインターフェースをライブラリ使用以前の状態に復帰する処理も行う．

1．初期化されていなければ何もしないで終了
2．保存しておいたRS-232Cハードウェア割り込み（INT 0Ch）の割り込みベクタを再設定する．
3．保存しておいた割り込みコントローラのマスク設定ポートおよびシステムポートの送受信許可状態を示すポートの内容を復帰する．

03：送信バッファに1バイト書き込む

【呼び出し方】　void rs_send_data(unsigned char)

送信すべきデータ（unsigned char型）を引き渡す．

【戻り値】

なし

【動作概要】

この関数を呼び出した時点ですぐにRS-232Cからデータを送出するわけではない．ここで行われるのは，送信のためのバッファにデータを登録することと，システムポートの送信許可状態を許可側にすることのみである．実際の送信処理はハードウェアが送信可能な状態となって発生する割り込みによって行われる．

また，この関数は送信バッファがいっぱいで登録できなかった場合でもエラーを返さない．必ず事前に「送信バッファ余裕バイト数取得」（後述）で送信バッファに空きがあるかどうかをチェックしてから呼び出すよう注意してほしい．

1．送信バッファに空きがあるかどうかチェックする．いっぱいになっていたら何もしないで終了する．
2．送信バッファデータセットポインタの示す位置にデータを格納する．格納後，ポインタを更新し，送信データカウンタを＋1する．
3．送信割り込み許可状態とする．送信割り込み許可状態とは，具体的にはシステムポートのTxEビットをオンにすることである．

パソコン通信用語集
CI信号　モデムから端末へ，モデムが着信信号を受信したことを知らせる信号（被呼信号検出）．

表6

04：受信バッファから1バイト取り出す

【呼び出し方】　unsigned char rs_receive_data(void)

【戻り値】

受信バッファから取り出したデータを unsigned char型で返す.

【動作概要】

この関数も，直接回線からデータを読み出して返すわけではない．割り込み処理によって，すでに受信バッファに格納されているデータを返すのである．したがって，実際の回線からの受信とアプリケーションがそのデータを引き取る処理は非同期で行われる．

当関数は，受信バッファにデータがなくてもエラーを返さない．回線からデータの受信があったかどうかは「受信済みデータ数取得」（後述）で知ることができるのでかならず事前にチェックしてから呼び出すこと．

また，当関数はデータを引き取ったあとの受信バッファの空き具合をチェックして，RS/CS制御およびXコントロールが有効な場合に残り容量の検査を行って，必要があればこれらの処理を行う．

1．受信データカウンタが0であれば，何もしないで終了する．
2．受信バッファデータセットポインタの示す位置からデータを取り出す．その後，ポインタを更新し，受信データカウンタを－1する．
3．RS制御で受信が停止状態（RSがオフの状態）であれば，受信バッファ空き容量をチェックして空きが指定点以上になっていればRSオンの処理を行い，受信処理が再開できるようにする．
4．Xコントロールで受信が停止状態であれば，受信バッファ空き容量をチェックして空きが指定点以上になっていれば，X-ON送信待ち状態として送信要求の処理を行う．

05：受信バッファから1バイト読み出す（ポインタは更新しない）

【呼び出し方】　unsigned char rs_peek_data(void)

【戻り値】

受信バッファから読み出したデータを unsigned char型で返す．

【動作概要】

これもすでに受信しているデータを読み出す機能をもった関数であるが，前出の「1バイト取り出し」と異なっているのは，受信バッファ先頭のデータを「覗く」だけで，実際に取り出さない点である．受信バッファのカウンタやポインタは変化しない．受信バッファの状態が変化しないので，RS制御やXコントロールでの状態遷移も当然発生しない．この関数も受信データがあるかどうかは「受信済みデータ数取得」（後述）で知ることができる．

1．受信データカウンタが0であれば，何もしないで終了する．
2．受信バッファデータセットポインタの示す位置からデータを読み出す．

06：送信バッファ余裕バイト数取得

【呼び出し方】　int rs_send_buffer_space(void)

【戻り値】

送信バッファの余裕バイト数を int型で返す．

【動作概要】

送信バッファにあと何バイト余裕があるかを返す．送信バッファ長から送信データカウンタを引いた値となる．

前述したように送信バッファへのデータ設定の際には，バッファがいっぱいでもエラーを返さない仕様になっている．アプリケーションプログラム側でかならず事前に当関数を使用して送信バッファに余裕があることを確認してから実際の送信処理を行うよう注意が必要である．

07：受信済みデータ数取得

【呼び出し方】　int rs_receive_count(void)

【戻り値】

受信済みデータのバイト数を int型で返す．

【動作概要】

受信バッファに何バイトのデータが格納されているかを返す．受信データカウンタの値となる．

前述したように受信バッファからデータを取り出す（または読み出す）関数ではバッファにデータがなくてもエラーを返さない仕様になっている．アプリケーションプログラム側でかならず事前に当関数を使用して受信バッファにデータがあることを確認してから実際の取り出し（読み出し）処理を行うよう注意が必要である．

08：送信待ちデータ数取得

【呼び出し方】　int rs_send_count(void)

【戻り値】

送信待ちデータ数を int型で返す．

【動作概要】

送信バッファに何バイトのデータが格納されているかを返す．送信データカウンタの値となる．

この関数の返す値は，アプリケーションプログラムが「送信バッファに1バイト書き込む」関数でライブラリに引き渡したデータのうち，実際に

表 6

回線に送出されていないデータ数である．引き渡したデータが実際に回線上に送られたかどうかのチェックに使用することになろう．

09：XコントロールモードON

【呼び出し方】　void rs_xcontrol_on(void)

【戻り値】
なし

【動作概要】
内部フラグをXコントロールONの状態にする．
この関数を呼び出した後，Xコントロールが有効状態となる．

10：XコントロールモードOFF

【呼び出し方】　void rs_xcontrol_off(void)

【戻り値】
なし

【動作概要】
内部フラグをXコントロールOFFの状態にする．
この関数を呼び出した後，Xコントロールが無効状態となる．

11：RTS（RS）ON

【呼び出し方】　void rs_rts_on(void)

【戻り値】
なし

【動作概要】
RS-232C用LSIのRSビットをオンにする出力を実行する．
前述したように，こちら側のRSビットをオンにすることによって，相手側のCSビットがオンになるような接続をしている場合に，この関数を呼び出すと相手側の送信が可能状態となり，データが送られてくるようになる．

12：RTS（RS）OFF

【呼び出し方】　void rs_rts_off(void)

【戻り値】
なし

【動作概要】
RS-232C用LSIのRTSビットをオフにする出力を実行する．
前述したように，こちら側のRSビットをオフにすることによって，相手側のCSビットがオフになるような接続をしている場合に，この関数を呼び出すと相手側の送信が不可能状態となり，データの送信(こちら側での受信)が停止するようになる．

13：DTR（ER）ON

【呼び出し方】　void rs_dtr_on(void)

【戻り値】
なし

【動作概要】
RS-232C用LSIのDTRビットをオンにする出力を実行する．
通常の接続の場合，こちら側のDTRをオンにすると，相手側のDSRがオンになる結果となる．

14：DTR（ER）OFF

【呼び出し方】　void rs_dtr_off(void)

【戻り値】
なし

【動作概要】
RS-232C用LSIのDTRビットをオフにする出力を実行する．
通常の接続の場合，こちら側のDTRをオフにすると，相手側のDSRがオフになる結果となる．

15：ブレーク信号ON

【呼び出し方】　void rs_break_on(void)

【戻り値】
なし

【動作概要】
RS-232C用LSIのSBRKビットをオンにする出力を実行する．

※注意
RS-232C用LSI（8251）のSBRKビットをオンにしている間，送信データの状態が強制的に“L”レベルに固定される．相手側でこの状態を検知してたとえば送信処理の強制中断などの処理を行う．パソコン通信を経験したことのある人なら「ブレーク送信」といったほうが早いかもしれない．

パソコン通信用語集
CS(CTS)信号　モデムから端末へデータの送信を要求する信号，ハードウェアフロー制御でRS信号と供に使用される(送信可)．

表6

Xコントロールが TYPEコマンドでの「CTRL＋S」「CTRL＋Q」にあたるものだとしたら，ブレーク送信は「CTRL＋C」に相当する。

16：ブレーク信号 OFF

【呼び出し方】　void rs_break_off(void)

【戻り値】
なし

【動作概要】
RS-232C用 LSIの SBRKビットをオフにする出力を実行する。
ブレーク信号は，一般に300ms程度の間 ON状態を継続する必要がある。呼び出し側でこの時間を監視して，当関数を呼び出すこと。「ブレーク信号ON」を呼び出したまま当関数を実行するのを忘れていると，ずっとブレーク信号ばかり出していることになるので注意してほしい。

17：Xコントロールモード検査

【呼び出し方】　int rs_xcontrol_check(void)

【戻り値】
以下の情報を int型で返す。

- XCON (=1)　：Xコントロール有効状態
- XCOFF(=0)　：Xコントロール無効状態

【動作概要】
現在，Xコントロールが有効であるか無効であるかを返す。

18：DCD (CD) 検査

【呼び出し方】　int rs_dcd_check(void)

【戻り値】
以下の情報を int型で返す。

- DCD ON　(=1)
- DCD OFF (=0)

【動作概要】
システムポートから DCDの状態を読んで返す。
通常の場合，DCD信号は通信相手との接続が行われているかどうかを示す。たとえば，ネットワークに接続していて，ホスト側で通信を切断した場合などにこの信号がオフになる（「ノーキャリア」などと通信ソフトで表示されるのは，こういう場合である）。

19：CTS (CS) 検査

【呼び出し方】　int rs_cts_check(void)

【戻り値】
以下の情報を int型で返す。

- CTS　ON　(=1)
- CTS　OFF (=0)

【動作概要】
システムポートから CTSの状態を読んで返す。
通常の場合，相手側が RTS信号を ON/OFF した場合に，こちら側の CTSが ON/OFFするように接続されている。98で使用している8251の場合，CTS信号がオフになるとハードウェア的に送信が不可の状態になる。

20：DSR (DR) 検査

【呼び出し方】　int rs_dsr_check(void)

【戻り値】
以下の情報を int型で返す。

- DSR　ON　(=1)
- DSR　OFF (=0)

【動作概要】
RS-232C用 LSIから DSRの状態を読んで返す。
通常の場合，相手側が DTR信号を ON/OFF した場合に，こちら側の DTRが ON/OFFするように接続されている。

21：タイムアウト監視用タイマ設定

【呼び出し方】　void rs_set_timer(int)
希望するタイムアウト時間（単位：10ms）を int型で引き渡す。

【戻り値】
なし

【動作概要】
タイマ BIOSの機能を使用して，指定した時間が経過したら割り込みがかかるように，設定を行う。指定するタイムアウト時間は10msを単位とする整数である。
この関数の中ではタイマ BIOSを以下のようにして呼び出している。

```
AH　　　<--- 02H
```

表6

CX　　　<--- タイムアウト時間（単位：10ms）
ES:BX　　<--- タイムアウト時の処理アドレス
INT 1Ch

タイムアウト時の処理アドレスには，指定した時間が経過したときにタイマ BIOSから呼び出されるルーチンのアドレスを指定する．このアドレスはソフト割り込みを使って呼び出されるため，IRETで終了する形にしておかなければならない．このタイムアウト時の処理ロジックは本ライブラリが内蔵する．

22：タイムアウトチェック

【呼び出し方】　int rs_chk_timer(void)

【戻り値】

以下の情報が int型で返される．

- 未タイムアウト　(=0)
- タイムアウト　(=1)

【動作概要】

rs_set_timer関数で設定したタイムアウト時間が経過したかどうかを問い合わせる関数である．本ライブラリのタイムアウト時処理ルーチン内でセットするフラグを参照して，タイムアウトが発生しているかどうかを返す．

リスト10
RS232LIB.ASM

```
            page 60,128
;==================================================================
;
;           ＲＳ－２３２Ｃライブラリ  (for PC-9801/286)
;
;                         Copyright(C) 1991 by Justech Labo.
;
;==================================================================
_TEXT     SEGMENT  WORD PUBLIC 'CODE'
_TEXT     ENDS
_DATA     SEGMENT  WORD PUBLIC 'DATA'
_DATA     ENDS
_BSS      SEGMENT  WORD PUBLIC 'BSS'
_BSS      ENDS

CGROUP   GROUP    _TEXT
DGROUP   GROUP    _BSS , _DATA

;//////////////////////////////////////////////////////////////////
;           定数定義
;//////////////////////////////////////////////////////////////////

;====== シリアルポート関係 ===========================
; ●ポート番号
USART_COM        EQU      32H      ; i8251 ｺﾏﾝﾄﾞ/ｽﾃｰﾀｽﾎﾟｰﾄ
USART_DAT        EQU      30H      ; i8251 ﾃﾞｰﾀﾎﾟｰﾄ

; ●各種コマンド
ASYNC_X16        EQU      02H      ; i8251 ﾓｰﾄﾞ設定ﾃﾞｰﾀ
USART_RESET      EQU      40H      ; i8251 リセットコマンド
RXE              EQU      04H      ; 受信ｲﾈｰﾌﾞﾙ  コマンド
TXE              EQU      01H      ; 送信ｲﾈｰﾌﾞﾙ  コマンド
RTS              EQU      20H      ; RTS on コマンド
DTR              EQU      02H      ; DTR on コマンド
RTS_RXE_DTR_TXE  EQU      RTS + RXE + DTR + TXE
ERR_RESET        EQU      10H      ; ｴﾗｰﾘｾｯﾄ  コマンド
SBRK             EQU      08H      ; ﾌﾞﾚｰｸ送出  コマンド

; ●各種ステータス
DSR              EQU      80H      ; DSR状態
ERR_STS          EQU      38H      ; エラー発生状態
TX_RDY           EQU      01H      ; 送信可状態
RX_RDY           EQU      02H      ; 受信可状態

;====== タイマポート関係 ============================
CTC_MODE         EQU      77H      ; ﾀｲﾏ ｺﾏﾝﾄﾞﾎﾟｰﾄ
CTC_COUNT2       EQU      75H      ; ﾎﾞｰﾚｰﾄ用ﾀｲﾏｶｳﾝﾀﾎﾟｰﾄ

CTC_C2_SET       EQU      0B6H     ; ﾀｲﾏｶｳﾝﾀﾛｰﾄﾞｺﾏﾝﾄﾞ

;====== システムポート関係 ===========================
PPI_W_C          EQU      37H      ; ｼｽﾃﾑﾎﾟｰﾄ ﾓｰﾄﾞ設定ﾎﾟｰﾄ
PPI_R_C          EQU      35H      ; i8255 port C read register
PPI_R_B          EQU      33H      ; i8255 port B read register

TXINT_GATE_ON    EQU      05H      ; PPI_W_C Tx int. gate on  command
TXINT_GATE_OFF   EQU      04H      ; PPI_W_C Tx int. gate off command
RXINT_GATE       EQU      01H      ; PPI_R_C Rx int. gate on  command
TXRXINT_GATE     EQU      05H      ; PPI_R_C Tx&Rx int. gate on  command
TXRXINT_MASK     EQU      NOT 07H  ; PPI_R_C Tx&Rx int. gate off command
CTS              EQU      40H      ; PPI_R_B CTS status
DCD              EQU      20H      ; PPI_R_B DCD status

;====== 割り込みコントローラ関係 =====================

PIC_OCW2         EQU      00H      ; 割り込みｺﾝﾄﾛｰﾗｺﾏﾝﾄﾞﾎﾟｰﾄ
PIC_IMR          EQU      02H      ; 割り込みｺﾝﾄﾛｰﾗﾏｽｸﾎﾟｰﾄ

EOI              EQU      20H      ; ＥＯＩコマンド
IMR_RS232C       EQU      10H      ; RS-232Cﾏｽｸﾊﾟﾀｰﾝ

;====== ＰＣ９８０１システムステータス ===============

SYS_SEGMENT      EQU      0000H    ; システムエリアのセグメント
SYS_OFFSET       EQU      0501H    ; BIOSﾌﾗｸﾞオフセット
SYS_CLOCK        EQU      80H      ; CPUｸﾛｯｸﾁｪｯｸﾃﾞｰﾀ

;====== 割り込みベクタ関係システムコール =============

INT_DOS          EQU      21H      ; システムコール
GET_INT          EQU      35H      ; 割り込みベクタ取得
SET_INT          EQU      25H      ; 割り込みベクタ設定
INT_8251         EQU      0CH      ; RS-232Cハード割り込み番号

;===== 送信バッファ ==================================

TXBUFFER_SIZE    EQU      400H
TXBUFFER_MASK    EQU      TXBUFFER_SIZE - 1

;===== 受信バッファ ==================================

RXBUFFER_SIZE    EQU      400H
RXBUFFER_MASK    EQU      RXBUFFER_SIZE - 1
ON_POINT         EQU      RXBUFFER_SIZE / 2
OFF_POINT        EQU      RXBUFFER_SIZE * 3 / 4

;====== Ｘコントロールキャラクタ =====================

X_ON             EQU      11H      ; (^Q)
X_OFF            EQU      13H      ; (^S)

;====== ブレーク信号送出時間 =========================

BREAK_TIME       EQU      50       ; 500msec

;====== その他 =======================================

ON               EQU      1
OFF              EQU      0

;====== 初期化用パラメータ設定値 =====================

;====== ボーレート定数 ======================
B75              EQU      0
B150             EQU      1
B300             EQU      2
B600             EQU      3
B1200            EQU      4
B2400            EQU      5
B4800            EQU      6
B9600            EQU      7
B19200           EQU      8

```


パソコン通信用語集
ER(DTR)信号　端末からモデムに通信可能状態になったことを知らせる信号(端末装置レディ)．

```
; ====== キャラクタ長定数 =================
L7              EQU     2
L8              EQU     3

; ====== パリティビット定数 ===============
PNONE           EQU     0
PODD            EQU     1
PEVEN           EQU     3

; ====== ストップビット長定数 =============
S1              EQU     1
S15             EQU     2
S2              EQU     3

; ====== X-ON/OFF定数 ====================
XCON            EQU     1
XCOFF           EQU     0

;//////////////////////////////////////////////////////////////
;        変数定義
;//////////////////////////////////////////////////////////////

_DATA   SEGMENT WORD PUBLIC 'DATA'

;====== ＳＩＯ初期化済みフラグ =====================

        PUBLIC  _rs_initialized

_rs_initialized DW      OFF

;====== ベクタ、マスク値など現状設定の退避 ==========

old_PPI         DB      ?
old_IMR         DB      ?
old_int_vect    DD      ?

usart_com_temp  DB      ?

;====== 送信リングバッファ管理用変数 ================

txbuffer        DB      TXBUFFER_SIZE DUP(?)
txbuffer_tail   DW      0
txbuffer_ptr    DW      0
txbuffer_counter DW     0

;====== 受信リングバッファ管理用変数 ================

rxbuffer        DB      RXBUFFER_SIZE DUP(?)
rxbuffer_tail   DW      0
rxbuffer_ptr    DW      0
rxbuffer_counter DW     0

;====== Ｘコントロール変数（受信用） ================

xcontrol_enable DW      XCOFF   ; XCOFF/XCON
xcontrol_stat   DW      OFF     ; OFF/ON
xcontrol_request DW     OFF     ; OFF/X_ON/X_OFF

;====== Ｘコントロール変数（送信用） ================

tx_enable       DW      ON      ; ON/OFF

;====== タイムアウト監視用ワーク ====================

time_flag_      DW      0

_DATA   ENDS

        PUBLIC  _rs_baud_rate, _rs_data_length, _rs_parity_bit
        PUBLIC  _rs_stop_bit, _rs_xcontrol

_BSS    SEGMENT WORD PUBLIC 'BSS'
;====== 初期化用パラメータ ==========================
; ●このエリアは外部から設定する必要があるため、PUBLIC宣言する。
_rs_baud_rate   DW      ?
_rs_data_length DW      ?
_rs_parity_bit  DW      ?
_rs_stop_bit    DW      ?
_rs_xcontrol    DW      ?
_BSS    ENDS

;//////////////////////////////////////////////////////////////
;        マクロ定義
;//////////////////////////////////////////////////////////////

;====== ｉ８２５１コマンド出力 ======================

out_dx_al       MACRO           ; entry dx:port
        out     dx,al
        REPT    6
        jmp     $ + 2
        ENDM
ENDM

out_dx          MACRO   REG     ; entry dx:port
        mov     al,REG
        out_dx_al
ENDM

;====== ｉ８２５１ステータス検査 ====================

usart_sts       MACRO   STS
        mov     dx,USART_COM
        in      al,dx
        test    al,STS
ENDM


;====== 割り込みベクタ取得 ==========================

dos_getvect     MACRO   VECT    ; return es+bx:address
        mov     ah,GET_INT
        mov     al,VECT
        int     INT_DOS
ENDM

;====== 割り込みベクタ設定 ==========================

dos_setvect     MACRO   VECT    ; entry bx+dx:address
        push    ds
        mov     ds,bx
        mov     ah,SET_INT
        mov     al,VECT
        int     INT_DOS
        pop     ds
ENDM

;//////////////////////////////////////////////////////////////
;        プログラム本体
;//////////////////////////////////////////////////////////////

_TEXT   SEGMENT WORD PUBLIC 'CODE'
        ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP

;==============================================================
;０１：ＲＳ２３２Ｃ初期化（通信使用開始）
;        【void rs_initialize(void)】
;==============================================================

        PUBLIC  _rs_initialize

_rs_initialize          PROC    NEAR
; 既に初期化済みであれば、いったん終了処理をおこなう。
        cmp     _rs_initialized,ON
        jne     rsini_1
        call    _rs_finish
rsini_1:
        push    dx
; 初期化済み状態とする
        mov     _rs_initialized,ON
;====== Ｘコントロール初期化 ==============
        call    initialize_xcontrol
        mov     tx_enable,ON
;====== 送受信バッファ初期化 ==============
        call    rec_buffer_flush
        call    send_buffer_flush
;====== オリジナルのレジスタ値を保存 =======
        mov     dx,PIC_IMR              ; 割り込みマスク
        in      al,dx
        mov     old_IMR,al
        mov     dx,PPI_R_C              ; 送受信許可状態
        in      al,dx
        mov     old_PPI,al
;====== ハードウエア設定 ==================
        cli
        call    stop_interrupt_51_to_59
        call    initialize_8253
        call    initialize_8251
        call    initialize_8259
;====== 割り込みベクタ設定 ================
        call    set_new_8251_vector
        call    start_interrupt_51_to_59
        sti
        pop     dx
        ret
_rs_initialize          ENDP

;==============================================================
;０２：通信使用終了
;        【void rs_finish(void)】
;==============================================================

        PUBLIC  _rs_finish

_rs_finish      PROC    NEAR
; 初期化がなされていなければ、なにもしない
        cmp     _rs_initialized,ON
        jne     rsfin_1
        push    dx
; 初期化済み状態を解除する
        mov     _rs_initialized,OFF
;====== 割り込みベクタ復帰 ================
        cli
        mov     di,offset old_int_vect
        mov     dx,[di]
        mov     bx,2[di]
        dos_setvect     INT_8251
;====== レジスタ復帰 ======================
        mov     dx,PPI_R_C
        mov     al,old_PPI
        out     dx,al
        mov     dx,PIC_IMR
        mov     al,old_IMR
        out     dx,al
        sti
        pop     dx
rsfin_1:
        ret
_rs_finish      ENDP

;==============================================================
;０３：送信バッファに一バイト書き込む
;        【void rs_send_data(unsigned char)】
;==============================================================

```

パソコン通信用語集

MNP　米Microcom社が提唱するデータ通信規約。エラーフリーやデータ圧縮の手順はモデム自身に内蔵されるのでアプリケーション側で意識する必要がない。

```
        PUBLIC  _rs_send_data

_rs_send_data   PROC    NEAR
        push    bp
        mov     bp,sp
        mov     ax,[bp+4]               ; 出力データを読み出す

        push    di
;====== バッファ余裕検査 ===================
        cmp     txbuffer_counter,TXBUFFER_SIZE
        jnc     send_1                  ; ﾊﾞｯﾌｧがいっぱいなら処理終了
;====== 送信データ書き込み ================
        mov     di,txbuffer_tail
        mov     txbuffer[di],al         ; ﾎﾟｲﾝﾀ位置にﾃﾞｰﾀを書き出す
        inc     di
        and     di,TXBUFFER_MASK
        mov     txbuffer_tail,di        ; ﾎﾟｲﾝﾀを更新
        cli
        inc     txbuffer_counter        ; ｶｳﾝﾀを更新
        sti
;====== 送信割り込み許可 ===================
        call    txint_enable
send_1:
        pop     di
        pop     bp
        ret
_rs_send_data   ENDP

;================================================================
;０４：受信バッファから一バイト取り出す
;       【unsigned char rs_receive_data(void)】
;================================================================

        PUBLIC  _rs_receive_data

_rs_receive_data        PROC    NEAR    ; retaun ax:data
        push    bx
        push    di
;====== 受信データ数検査 ===============
        xor     bx,bx
        cmp     rxbuffer_counter,0
        je      reci_2                  ; 受信ﾃﾞｰﾀがなければ終わり
;====== 受信データ取り出し ============
        mov     di,rxbuffer_ptr
        mov     bl,rxbuffer[di]         ; ﾎﾟｲﾝﾀ位置から読み出す
        inc     di
        and     di,RXBUFFER_MASK
        mov     rxbuffer_ptr,di         ; ﾎﾟｲﾝﾀを更新
        cli
        dec     rxbuffer_counter        ; ｶｳﾝﾀを更新
        sti
;====== 受信余裕検査（ＲＳコントロール） ===
        test    usart_com_temp,RTS
        jnz     reci_1
        cmp     rxbuffer_counter,ON_POINT
        jnc     reci_1
; バッファに余裕ができたので、ＲＴＳをオンして受信可能とする
        call    _rs_rts_on
;====== 受信余裕検査（Ｘコントロール） =====
reci_1:
        cmp     xcontrol_stat,ON
        jne     reci_2                  ; Xｺﾝﾄﾛｰﾙが無効状態
        cmp     rxbuffer_counter,ON_POINT
        jnc     reci_2
; バッファに余裕ができたので、Ｘ－ＯＮ状態を送信できる
        mov     xcontrol_request,X_ON
        call    txint_enable
reci_2:
        mov     ax,bx
        pop     di
        pop     bx
        ret
_rs_receive_data        ENDP

;================================================================
;０５：受信ﾊﾞｯﾌｧから一バイト読み出す（ポインタは更新しない）
;       【unsigned char rs_peek_data(void)】
;================================================================

        PUBLIC  _rs_peek_data

_rs_peek_data   PROC    NEAR    ; return ax:data
        xor     ax,ax
        cmp     rxbuffer_counter,0
        je      rspeek_1                ; 受信ﾃﾞｰﾀがなければ終わり
        push    di
        mov     di,rxbuffer_ptr
        mov     al,rxbuffer[di]         ; ﾎﾟｲﾝﾀ位置から読み出す
        pop     di
rspeek_1:
        ret
_rs_peek_data   ENDP

;================================================================
;０６：送信ﾊﾞｯﾌｧ余裕バイト数取得
;       【int rs_send_buffer_space(void)】
;================================================================

        PUBLIC  _rs_send_buffer_space

_rs_send_buffer_space   PROC    NEAR    ; return ax:size
        mov     ax,TXBUFFER_SIZE
        sub     ax,txbuffer_counter     ; ﾊﾞｯﾌｧｻｲｽﾞ - 送信待ちﾃﾞｰﾀ数
        ret
_rs_send_buffer_space   ENDP

;================================================================
;０７：受信済みデータ数取得
;       【int rs_receive_count(void)】
;================================================================

        PUBLIC  _rs_receive_count

_rs_receive_count       PROC    NEAR    ; return ax:count
        mov     ax,rxbuffer_counter
        ret
_rs_receive_count       ENDP

;================================================================
;０８：送信待ちデータ数取得
;       【int rs_send_count(void)】
;================================================================

        PUBLIC  _rs_send_count

_rs_send_count          PROC    NEAR    ; return ax:count
        mov     ax,txbuffer_counter
        ret
_rs_send_count          ENDP

;================================================================
;０９：ＸコントロールモードＯＮ
;       【void rs_xcontrol_on(void)】
;================================================================

        PUBLIC  _rs_xcontrol_on

_rs_xcontrol_on         PROC    NEAR
        mov     _rs_xcontrol,XCON
        call    initialize_xcontrol
        ret
_rs_xcontrol_on         ENDP

;================================================================
;１０：ＸコントロールモードＯＦＦ
;       【void rs_xcontrol_off(void)】
;================================================================

        PUBLIC  _rs_xcontrol_off

_rs_xcontrol_off        PROC    NEAR
        mov     _rs_xcontrol,XCOFF
        call    initialize_xcontrol
        ret
_rs_xcontrol_off        ENDP

;================================================================
;１１：ＲＴＳ（ＲＳ）ＯＮ
;       【void rs_rts_on(void)】
;================================================================

        PUBLIC  _rs_rts_on

_rs_rts_on      PROC    NEAR
        push    dx
        mov     dx,USART_COM
        mov     al,usart_com_temp
        or      al,RTS                          ; 他のビットを変えないように
        mov     usart_com_temp,al
        out_dx_al
        pop     dx
        ret
_rs_rts_on      ENDP

;================================================================
;１２：ＲＴＳ（ＲＳ）ＯＦＦ
;       【void rs_rts_off(void)】
;================================================================

        PUBLIC  _rs_rts_off

_rs_rts_off     PROC    NEAR
        push    dx
        mov     dx,USART_COM
        mov     al,usart_com_temp
        and     al,NOT RTS                      ; 他のビットを変えないように
        mov     usart_com_temp,al
        out_dx_al
        pop     dx
        ret
_rs_rts_off     ENDP

;================================================================
;１３：ＤＴＲ（ＥＲ）ＯＮ
;       【void rs_dtr_on(void)】
;================================================================

        PUBLIC  _rs_dtr_on

_rs_dtr_on      PROC    NEAR
        push    dx
        mov     dx,USART_COM
        mov     al,usart_com_temp
        or      al,DTR                          ; 他のビットを変えないように
        mov     usart_com_temp,al
        out_dx_al
        pop     dx
        ret
_rs_dtr_on      ENDP

;================================================================
;１４：ＤＴＲ（ＥＲ）ＯＦＦ
;       【void rs_dtr_off(void)】
;================================================================
```

パソコン通信用語集

MNPクラス４　モデム間は同期通信を行い，回線状況によってデータのパケットサイズを変更する．スタートビットとストップビットがなくなりスループット120%以上．

```

        PUBLIC  _rs_dtr_off

_rs_dtr_off     PROC    NEAR
        push    dx
        mov     dx,USART_COM
        mov     al,usart_com_temp
        and     al,NOT DTR                      ; 他のビットを変えないように
        mov     usart_com_temp,al
        out_dx_al
        pop     dx
        ret
_rs_dtr_off     ENDP

;=====================================================================
;１５：ブレーク信号ＯＮ
;       【void rs_break_on(void)】
;=====================================================================

        PUBLIC  _rs_break_on

_rs_break_on    PROC    NEAR
        push    dx
;====== 送信バッファクリア ================
        call    send_buffer_flush
        call    txint_disable
;====== ブレークオン =====================
        mov     dx,USART_COM
        mov     al,usart_com_temp
        or      al,SBRK                         ; 他のビットを変えないように
        mov     usart_com_temp,al
        out_dx_al
        pop     dx
        ret
_rs_break_on    ENDP

;=====================================================================
;１６：ブレーク信号ＯＦＦ
;       【void rs_break_off(void)】
;=====================================================================

        PUBLIC  _rs_break_off

_rs_break_off   PROC    NEAR
        push    dx
        mov     dx,USART_COM
        mov     al,usart_com_temp
        and     al,NOT SBRK                     ; 他のビットを変えないように
        mov     usart_com_temp,al
        out_dx_al
        call    txint_enable
        pop     dx
        ret
_rs_break_off   ENDP

;=====================================================================
;１７：Ｘコントロールモード検査
;       【int rs_xcontrol_check(void)】
;=====================================================================

        PUBLIC  _rs_xcontrol_check

_rs_xcontrol_check      PROC    NEAR    ; exit ax:XCON/XCOFF
        mov     ax,_rs_xcontrol
        ret
_rs_xcontrol_check      ENDP


;=====================================================================
;１８：ＤＣＤ（ＣＤ）検査
;       【int rs_dcd_check(void)】
;=====================================================================

        PUBLIC  _rs_dcd_check

_rs_dcd_check   PROC    NEAR    ; return ax:ON/OFF
        push    dx
        mov     dx,PPI_R_B
        in      al,dx
        not     al                              ; 他のビットを変えないように
        and     ax,DCD
        jz      dcdchk_1
        mov     ax,ON
dcdchk_1:
        pop     dx
        ret
_rs_dcd_check   ENDP

;=====================================================================
;１９：ＣＴＳ（ＣＳ）検査
;       【int rs_cts_check(void)】
;=====================================================================

        PUBLIC  _rs_cts_check

_rs_cts_check   PROC    NEAR    ; return ax:ON/OFF
        push    dx
        mov     dx,PPI_R_B
        in      al,dx
        not     al                              ; 他のビットを変えないように
        and     ax,CTS
        jz      ctschk_1
        mov     ax,ON
ctschk_1:
        pop     dx
        ret
_rs_cts_check   ENDP

;=====================================================================
;２０：ＤＳＲ（ＤＲ）検査
;       【int rs_dsr_check(void)】
;=====================================================================

        PUBLIC  _rs_dsr_check

_rs_dsr_check   PROC    NEAR    ; return ax:ON/OFF
        push    dx
        mov     dx,USART_COM
        in      al,dx
        and     ax,DSR                          ; 他のビットを変えないように
        jz      dsrchk_1
        mov     ax,ON
dsrchk_1:
        pop     dx
        ret
_rs_dsr_check   ENDP

;=====================================================================
;２１：タイムアウト監視用タイマ設定
;       【void rs_set_timer(int)】
;=====================================================================

        PUBLIC  _rs_set_timer

_rs_set_timer           PROC    NEAR
        mov     time_flag_,0
        push    bp
        mov     bp,sp
        push    cx
        push    es
        push    bx
        mov     cx,[bp+4]                       ; 設定時間を読み出す
        mov     ah,2
        push    cs
        pop     es
        mov     bx,offset tmintpc_              ; ﾀｲﾑｱｳﾄ処理ｱﾄﾞﾚｽ
        int     1ch                             ; ﾀｲﾏBIOSの実行
        pop     bx
        pop     es
        pop     cx
        pop     bp
        ret
_rs_set_timer           ENDP

;=====================================================================
;２２：タイムアウトチェック
;       【bool rs_chk_timer(void)】
;=====================================================================

        PUBLIC  _rs_chk_timer

_rs_chk_timer           PROC    NEAR
        xor     ax,ax
        cmp     time_flag_,0
        jz      _rs_chk_timer_not
        inc     ax
_rs_chk_timer_not:
        ret
_rs_chk_timer           ENDP

;=====================================================================
;       ボーレート設定（８２５３イニシャライズ）
;=====================================================================

;====== ボーレート設定用定数テーブル ================

baud_table_10MHz        DW      2048,1024,512,256,128,64,32,16,8
baud_table_08MHz        DW      1664, 832,416,208,104,52,26,13,6

initialize_8253 PROC    NEAR
        push    bx
        push    dx
        push    di
        push    es
;====== ボーレート ========================
        mov     bx,_rs_baud_rate
        shl     bx,1
;====== クロック検査 =====================
        mov     ax,SYS_SEGMENT
        mov     es,ax
        mov     di,SYS_OFFSET
        mov     al,es:[di]
        test    al,SYS_CLOCK
        jnz     i8253_1
        mov     bx,baud_table_10MHz[bx];
        jmp     i8253_2
i8253_1:
        mov     bx,baud_table_08MHz[bx];
;====== カウンタ設定 =====================
i8253_2:
        mov     dx,CTC_MODE
        out_dx  CTC_C2_SET
        mov     dx,CTC_COUNT2
        out_dx  bl
        out_dx  bh
        pop     es
        pop     di
        pop     dx
        pop     bx
        ret
initialize_8253 ENDP

;=====================================================================
;       通信データフォーマット設定（８２５１イニシャライズ）
;=====================================================================
```

パソコン通信用語集

MNPクラス5　MNPクラス4に加え，データ圧縮が行われる．これによってスループットは最大200%となる．

```
initialize_8251 PROC    NEAR
        push    cx
        push    dx
;====== 非同期式　１／１６ ================
        mov     ch,ASYNC_X16
;====== ストップビット数 =================
        mov     ax,_rs_stop_bit
        and     al,03H
        ror     al,1
        ror     al,1
        or      ch,al
;====== パリティ指定 ======================
        mov     ax,_rs_parity_bit
        and     al,03H
        mov     cl,4
        shl     al,cl
        or      ch,al
;====== データビット数 ====================
        mov     ax,_rs_data_length
        and     al,03H
        shl     al,1
        shl     al,1
        or      ch,al
;====== ８２５１リセット =================
        mov     dx,USART_COM
        xor     al,al
        REPT    3
        out_dx_al
        ENDM
        out_dx  USART_RESET
;====== ８２５１モード設定 ================
        out_dx  ch
;====== ８２５１コマンド設定 ==============
        mov     al,RTS_RXE_DTR_TXE
        mov     usart_com_temp,al
        out_dx_al
;====== ダミーリード =====================
        mov     dx,USART_DAT
        in      al,dx
        pop     dx
        pop     cx
        ret
initialize_8251 ENDP

;==============================================================
;       ８２５１割り込みベクタ設定
;==============================================================

set_new_8251_vector     PROC    NEAR
        push    bx
        push    dx
        push    di
        push    es
;====== 元のアドレスを保存 ================
        dos_getvect     INT_8251
        mov     di,offset old_int_vect
        mov     [di],bx
        mov     2[di],es
;====== 割り込みアドレス設定 ==============
        mov     dx,offset rsintrp
        mov     bx,_TEXT
        dos_setvect     INT_8251
        pop     es
        pop     di
        pop     dx
        pop     bx
        ret
set_new_8251_vector     ENDP

;==============================================================
;       ８２５１からの送受信割り込み禁止
;       現在の状態は保存済み
;==============================================================

stop_interrupt_51_to_59         PROC    NEAR
        push    dx
        mov     al,old_PPI
        and     al,TXRXINT_MASK
        out     dx,al
        pop     dx
        ret
stop_interrupt_51_to_59         ENDP

;==============================================================
;       ８２５１からの受信割り込み許可
;       送信割り込みは送信時に許可にする
;==============================================================

start_interrupt_51_to_59        PROC    NEAR
        push    dx
        mov     dx,PPI_R_C
        in      al,dx
        jmp     $ + 2
        or      al,RXINT_GATE
        out     dx,al
        pop     dx
        ret
start_interrupt_51_to_59        ENDP

;==============================================================
;       ８２５９のＲＳ２３２Ｃ割り込み許可
;       現在の状態は保存済み
;==============================================================

initialize_8259         PROC    NEAR
        push    dx
        mov     dx,PIC_IMR
        in      al,dx
        and     al,NOT IMR_RS232C
        out     dx,al
        pop     dx
        ret
initialize_8259         ENDP

;==============================================================
;       Ｘコントロール機能の初期化
;==============================================================

initialize_xcontrol     PROC    NEAR
        mov     ax,_rs_xcontrol
        mov     xcontrol_enable,ax
        mov     xcontrol_stat,OFF
        mov     xcontrol_request,OFF
        ret
initialize_xcontrol     ENDP

;==============================================================
;       送信割り込み許可、禁止
;==============================================================

txint_enable    PROC    NEAR
        push    dx
        mov     dx,PPI_W_C
        mov     al,TXINT_GATE_ON
        out     dx,al
        jmp     $ + 2
        pop     dx
        ret
txint_enable    ENDP

txint_disable   PROC    NEAR
        push    dx
        mov     dx,PPI_W_C
        mov     al,TXINT_GATE_OFF
        out     dx,al
        jmp     $ + 2
        pop     dx
        ret
txint_disable   ENDP

;==============================================================
;       送信バッファの初期化
;==============================================================

;====== 受信バッファクリア ========================

rec_buffer_flush        PROC    NEAR
        xor     ax,ax
        mov     rxbuffer_tail,ax
        mov     rxbuffer_ptr,ax
        mov     rxbuffer_counter,ax
        ret
rec_buffer_flush        ENDP

;====== 送信バッファクリア ========================

send_buffer_flush       PROC    NEAR
        xor     ax,ax
        mov     txbuffer_tail,ax
        mov     txbuffer_ptr,ax
        mov     txbuffer_counter,ax
        ret
send_buffer_flush       ENDP

;==============================================================
;       送受信ハードウェア割り込み処理
;==============================================================

rsintrp         PROC    FAR
        push    ax
        push    dx
        push    ds
        mov     ax,DGROUP
        mov     ds,ax
        cld
;====== 受信割り込み検査 ==================
rsint_1:
        usart_sts RX_RDY
        jz      rsint_2
        call    rsint_rx
        usart_sts TX_RDY
        jz      rsint_3
        call    rsint_tx
        jmp     rsint_3
;====== 送信割り込み検査 ==================
rsint_2:
        test    al,TX_RDY
        jz      rsint_3
        call    rsint_tx
        usart_sts RX_RDY
        jz      rsint_3
        call    rsint_rx
;====== ＥＯＩ出力 ========================
rsint_3:
        mov     dx,PIC_OCW2
        mov     al,EOI
        out     dx,al
        pop     ds
        pop     dx
        pop     ax
        iret
rsintrp         ENDP
```


パソコン通信用語集
MNPクラス７　MNPクラス５のデータ圧縮機能を強化している．直前のデータから次のデータを予測し，より有効な圧縮を行う．スループットは最大300%となる．

```
;====== 送信割り込み処理 ============================

rsint_tx        PROC    NEAR
;====== Ｘコントロール検査 =================
        cmp     xcontrol_request,X_OFF
        jne     rstx_1
        mov     xcontrol_stat,ON
        mov     xcontrol_request,OFF
        mov     dx,USART_DAT
        out_dx  X_OFF
        ret
rstx_1:
        cmp     xcontrol_request,X_ON
        jne     rstx_2
        mov     xcontrol_stat,OFF
        mov     xcontrol_request,OFF
        mov     dx,USART_DAT
        out_dx  X_ON
        ret
;====== 送信データ数検査 ==================
rstx_2:
        cmp     txbuffer_counter,0
        je      rstx_3
;====== 送信抑止中検査 ====================
        cmp     tx_enable,OFF
        jne     rstx_4
;====== 送信割り込み禁止 ==================
rstx_3:
        call    txint_disable
        ret
;====== バッファから一バイト取り出す =======
rstx_4:
        push    di
        mov     di,txbuffer_ptr
        mov     al,txbuffer[di]
        inc     di
        and     di,TXBUFFER_MASK
        mov     txbuffer_ptr,di
        dec     txbuffer_counter
;====== データ送信 ========================
        mov     dx,USART_DAT
        out_dx_al
        pop     di
        ret
rsint_tx        ENDP

;====== 受信割り込み処理 ============================

rsint_rx        PROC    NEAR    ; entry al:i8251 status
;====== 受信エラー検査 ====================
        test    al,ERR_STS
        jz      rsrx_1
;====== エラーリセット ====================
        mov     al,usart_com_temp
        or      al,ERR_RESET
        out_dx_al
        mov     dx,USART_DAT
        in      al,dx           ; dummy read
        ret
;====== 受信データ入力 ====================
rsrx_1:
        mov     dx,USART_DAT
        in      al,dx
;====== Ｘコントロール処理 =================
        cmp     xcontrol_enable,XCON
        jne     rsrx_3
;====== ＸＯＦＦ受信検査 =========
        cmp     al,X_OFF
        jne     rsrx_2
        mov     tx_enable,OFF
        ret
;====== ＸＯＮ受信検査 ===========
rsrx_2:
        cmp     al,X_ON
        jne     rsrx_3
        mov     tx_enable,ON
        call    txint_enable
        ret
;====== 受信バッファオーバフロー検査 =======
rsrx_3:
        cmp     rxbuffer_counter,RXBUFFER_SIZE
        jc      rsrx_4
        ret
;====== データ書き込み ====================
rsrx_4:
        push    di
        mov     di,rxbuffer_tail
        mov     rxbuffer[di],al
        inc     di
        and     di,RXBUFFER_MASK
        mov     rxbuffer_tail,di
        inc     rxbuffer_counter
;====== 受信余裕検査（ＲＳコントロール） ===
        cmp     rxbuffer_counter,OFF_POINT
        jc      rsrx_5
        call    _rs_rts_off
;====== 受信余裕検査（Ｘコントロール） =====
        cmp     xcontrol_enable,XCON
        jne     rsrx_5
        mov     xcontrol_request,X_OFF
        call    txint_enable
rsrx_5:
        pop     di
        ret
rsint_rx        ENDP

;=================================================================
;       タイマ割り込み処理
;=================================================================
; ※「_rs_set_timer」で設定された時間が経過すると、タイマＢＩＯＳから
;   このロジックがコールされる。time_flag_に－１をセットしておいて
;  「_rs_chk_timer」からチェックされるのを待つ。
tmintpc_        PROC    FAR
        push    ds
        push    ax
        mov     ax,DGROUP
        mov     ds,ax
        mov     time_flag_,0FFFFH
        pop     ax
        pop     ds
        iret
tmintpc_        ENDP

_TEXT   ENDS
        END
```

以降は直接アプリケーションとは関係ないが，ライブラリ全体の動作を知るうえで参考になる実際の送受信割り込み動作とタイマ動作の概要である．

送受信ハード割り込み処理概要

本ライブラリでは，通信処理を送信・受信ともに割り込み処理で行っている．そのため，割り込み処理(ハードウェア割り込みINT 0Ch)はRS-232Cからデータを受信したとき，および送信可能な状態になったときに処理を開始する．注意すべきなのは，送信・受信と要因の異なる割り込み処理が同じINT番号で開始されることで，処理開始時に「この割り込みは受信で発生したのか？　送信で発生したのか？」という判定をしなければならない．さらに，受信で発生した割り込みを処理している最中に今度は送信での割り込み要求が発生しているかもしれないという可能性がある(逆のこともいえる)．

そこで，受信処理（あるいは送信処理)が終了したときに，再度送信処理(あるいは受信処理)が発生していないかどうかをチェックしている．具体的な割り込み処理手順は図16のとおりである．

【送信処理概要】

1．X-OFF要求状態であれば，X-OFF(13h)を送信して処理を終了する．
2．X-ON要求状態であれば，X-ON (11h) を送信して処理を終了する．
3．送信カウンタが0または送信状態フラグがOFFであれば，送信割り込みを禁止状態にして処理を終了する．

パソコン通信用語集
PDS (Public Domain Software)　著作者が著作物を広く利用してもらうために著作権を放棄したソフトのこと．「PDS」に分類されるソフトは日本にはほとんどない．

4．送信バッファのポインタの示す位置からデータを取り出し，ポインタおよびカウンタの更新を行う．
5．取り出したデータを RS-232C 用 LSI に送出して処理を終了する．

【受信処理概要】

1．受信エラーが発生していれば，エラーリセットを行った後，ダミーの読み出しをして処理を終了する．
2．RS-232C用LSIから1バイト受信する．
3．Xコントロールが有効であれば以下の処理を行う．
　・受信したデータが X-OFF(13h)であれば送信状態フラグを OFF にして処理を終了する．
　・受信したデータが X-ON(11h)であれば送信状態フラグを ON にした後，送信を許可状態にして処理を終了する．
4．受信バッファがいっぱいになっていたら処理を終了する．
5．受信バッファのポインタの示す位置へデータを格納し，ポインタおよびカウンタの更新を行う．
6．受信バッファの残量が設定値を超えたら以下の処理を行う．
　・RTS (RS) OFF を行う．
　・Xコントロールが有効であれば，X-OFF 要求状態として送信許可にする．

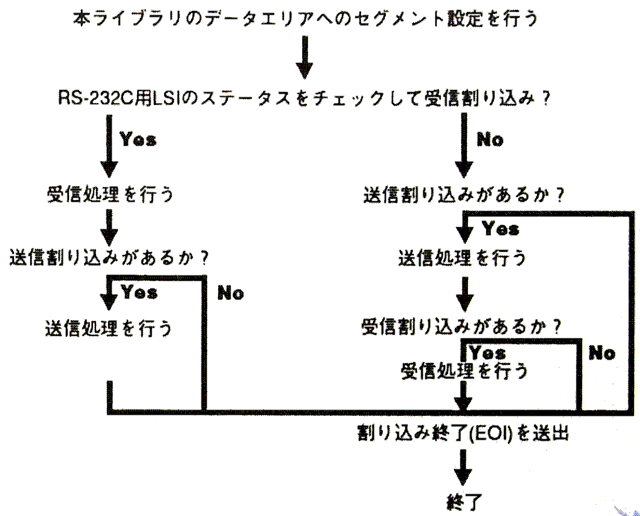


図16

タイムアウト割り込み処理概要

タイムアウト監視のためにタイマ BIOS を使用する．

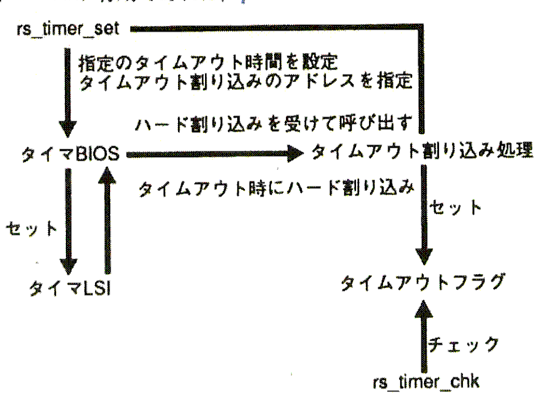


図17

アセンブルについて

今回の通信ライブラリの開発には，Borland 社の Borland　C++ Version 2.00 に含まれる「Turbo　Assembler　Version 2.51」を使用した．

```
A>TASM /ML RS232LIB;
```

とすればアセンブルできる．「/ML」オプションは，C言語とリンクするためにシンボルを小文字のままオブジェクトにするためである．このオプションをつけないと，シンボルが大文字でアセンブルされてしまうため，リンク時点で参照できなくなるので注意してほしい．

パソコン通信用語集
RS(RTS)信号　端末からモデムへデータの送信を要求する信号，ハードウェアフロー制御で CS信号と供に使用される (送信要求)．

呼び出し例とリンク方法

Cで記述されたプログラムからライブラリの関数を使用するためのヘッダファイルをリスト11に示す．後半の初期化用定数は，「rs_initialize」を呼び出す前にエリアに設定すべき数値の定義である．アプリケーションプログラムは，このヘッダファイルをインクルードしたうえでコンパイルすること．

では，具体的にC言語で記述したプログラムから通信ライブラリを使用する例をあげてみよう．次章でこのライブラリを使用した通信プログラムを紹介するが，ここではサンプル関数としていくつかの例を出してみる．

リスト12は，アプリケーションプログラムでこの通信ライブラリを使う場合に必須な初期化処理の呼び出しである．「rs_initialize」関数は，通信ライブラリ内にもっている初期化用変数を呼び出し側からセットしたうえで実行する必要があるので，このようにパラメータとして通信条件を受け取ってエリアにセットして呼び出す関数を挿入するのも1手法かもしれない．

リスト13は1バイト受信の関数であるが，受信データがあった場合にはそのデータを，ないときは一定時間待ち，時間内に受信がなければ「-1」を返す．

まず，「rs_receive_count」で受信データがあるかどうかをチェックする．0以外が返ってきたときはデータがあるので，すぐに「rs_receive_data」を

LIST

リスト11　RS232LIB.H

```
void    rs_initialize(void);
void    rs_finish(void);
void    rs_send_data(unsigned char);
unsigned char   rs_receive_data(void);
unsigned char   rs_peek_data(void);
int     rs_send_buffer_space(void);
int     rs_receive_count(void);
int     rs_send_count(void);
void    rs_xcontrol_on(void);
void    rs_xcontrol_off(void);
void    rs_rts_on(void);
void    rs_rts_off(void);
void    rs_dtr_on(void);
void    rs_dtr_off(void);
void    rs_break_on(void);
void    rs_break_off(void);
int     rs_xcontrol_check(void);
int     rs_dcd_check(void);
int     rs_cts_check(void);
int     rs_dsr_check(void);
void    rs_set_timer(int);
int rs_chk_timer(void);

extern  int rs_baud_rate, rs_data_length, rs_parity_bit;
extern  int rs_stop_bit, rs_xcontrol,    rs_initialized;

/* ====== ボーレート定数 ====================== */

#define     B75             0
#define     B150            1
#define     B300            2
#define     B600            3
#define     B1200           4
#define     B2400           5
#define     B4800           6
#define     B9600           7
#define     B19200          8

/* ====== キャラクタ長定数 ================= */
#define     L7              2
#define     L8              3

/* ====== パリティビット定数 =============== */
#define     PNONE           0
#define     PODD            1
#define     PEVEN           3

/* ====== ストップビット長定数 ============= */
#define     S1              1
#define     S15             2
#define     S2              3

/* ====== X-ON/OFF定数 ==================== */
#define     XCON            1
#define     XCOFF           0
```

リスト12

```
/* ========================================
 *  RS232C関係初期化
 * ======================================== */

int RS_open(int baud, int data, int parity, int stop,
            int xctrl, int port_number, int mode)
{
    rs_baud_rate    = baud;
    rs_data_length  = data;
    rs_parity_bit   = parity;
    rs_stop_bit     = stop;
    rs_xcontrol     = xctrl;

    rs_initialize();

    return  TRUE;
}
```

パソコン通信用語集

RS-232C　米EIAがCCITT V.24, V.28勧告に従って定めたシリアルインターフェース規格．信号線の電気的仕様，信号線の種類と機能などの機械的仕様からなる．

実行してその戻り値をそのまま返す．データがない場合は「rs_set_timer」を使ってタイムアウト時間を設定して（rs_set_timerでの設定時間は10ms単位であるので，5000をセットすると50秒ということになる）データ数チェックを繰り返す．このループの脱出条件は，受信データ数が0以外になって受信したデータを引き取ったとき，またはrs_chk_timerによってタイムアウトが認識されたときである．

通信ライブラリのデータ送信は1バイトデータを送信するものである．リスト14の関数はこれを連続して繰り返して指定された文字列を送信する．この関数に引き渡す文字列は当然ヌルターミネイトされていなければならない．

リスト15の関数は改行コード(0Dh)を1バイト送信するためのものである．単機能であるが，使用頻度は高いので用意しておいて損はないと思う．

前出の2つの関数を組み合わせて，指定の文字列に改行を加えた「1行」を送信する関数である(リスト16)．

リスト17は，ブレーク送出のための関数である．ブレーク状態の保持時間は処理系によっても微妙に異なるが，一般的に300ms程度必要となる．「rs_break_on」でブレーク状態にした後，「rs_set_timer」で300msのタイムアウトを設定してタイムアウトを待つ．ループから抜けてきたところで「rs_break_off」でブレーク状態の解除を行う．持続時間は外部からパラメータとして与えるようにしてもよいだろう．

リスト18は，RS信号およびER信号のON/OFF関数である．ライブラリではこれらの信号制御はすべてON/OFFいずれかの単機能になっているので，こうして外部からスイッチを受け取ってON/OFFどちらかを実

LIST

リスト13

```
/* ========================================
 *  タイムアウト監視付き１バイト受信
 * ======================================== */

int     rtimer = 5000;          /* 受信タイムアウト値 (*10msec) */

int     RS_receive_wait(void)
{
    if(rs_receive_count()) return(rs_receive_data());

    rs_set_timer(rtimer);
    for(;;)
    {
        if(rs_receive_count())      return(rs_receive_data());
        if(rs_chk_timer())          return -1;
    }
}
```

リスト14

```
/* ========================================
 *  文字列出力
 * ======================================== */

void    RS_send_string(char *ptr)
{
    for(; *ptr ; ptr++)
        rs_send_data(*ptr);
}
```

リスト15

```
/* ========================================
 *  改行送信
 * ======================================== */

void        RS_send_cr(void)
{
    rs_send_data(0x0d);
}
```

リスト16

```
/* ========================================
 *  一行出力
 * ======================================== */

void    RS_send_line(char *ptr)
{
    RS_send_string(ptr);
    RS_send_cr();
}
```

リスト17

```
/* ========================================
 *  ブレーク信号送出
 * ======================================== */

static  int     break_time = 30;     /* 300ms */

void    RS_send_break(void)
{
    rs_break_on();

    rs_set_timer(break_time);
    while(!rs_chk_timer());

    rs_break_off();
}
```


パソコン通信用語集
X-MODEM　無手順でバイナリファイルなどを送るためのファイル転送手順．ファイルを128バイト単位のブロックで送り，受信側で確認を繰り返す．

行する関数を作成することにしてもよい．

リスト19は，Xコントロールの有効／無効を切り替える関数である．信号線の制御と同様にライブラリではON/OFFの単機能となっているので，どちらかを実行する関数を例出してみた．

最後は「指定された時間だけ送信バッファが空になるのを待つ」関数である（リスト20）．指定された時間が経過したかどうかをチェックするために「rs_set_timer」と「rs_chk_timer」を使用する．「rs_set_timer」で設定した時間が経過すると「rs_chk_timer」が真を返してくるので，それまでの間「rs_send_count」を呼び続ける．タイムアウト前に「rs_send_count」が0を返してきたらこの関数の戻り値が真となる．

ライブラリ関数の使い方の例を実際にコーディングしてみたが，あらましはご理解いただけたであろうか．受信・送信の際にタイムアウト監視つきのループを作って，その中で受信待ち，送信待ちをするのが勘どころである．受信したデータは，画面に表示するなり，ファイルに書き込むなり，ここからあとは通常のプログラムとかわりないので安心してほしい．

LIST

リスト18

```
/* ========================================
 *  RS, ERのON／OFF  (0でOFF)
 * ======================================== */

void    RS_rs_set(int mode)
{
    if(mode == OFF)
        rs_rts_off();
    else
        rs_rts_on();
}

void    RS_er_set(int mode)
{
    if(mode == OFF)
        rs_dtr_off();
    else
        rs_dtr_on();
}
```

リスト19

```
/* ========================================
 *  Xコントロールモードの設定
 * ======================================== */

void    RS_Xcontrol_set(int xctrl)
{
    if(xctrl == OFF)
        rs_xcontrol_off();
    else
        rs_xcontrol_on();
}
```

リスト20

```
/* ======================================================
 *  指定した時間の間、送信バッファが空になるのを待つ
 * ====================================================== */

int     RS_send_buffer_empty(int wait_count)
{
    rs_set_timer(wait_count);
    do
    {
        if(!rs_send_count())     return TRUE;
    }while(!rs_chk_timer());
    return  FALSE;
}
```

通信ライブラリ編の終りに

以上で，通信ライブラリ編は終りである．次章ではいよいよこのライブラリを使った具体的な通信ソフト（今回は，柔軟な動作指定をユーザーが作り出せる通信言語という形態を取ってみた）を作っていくことになるが，それ以外でもオリジナルな通信ソフトを作るときの手助けになれば幸いである．

通信ソフトというと何か特殊な技術やノウハウが必要というイメージもあるが，ものすごく高性能とか高機能を目指さないかぎり，この通信ライブラリの関数を使ってプログラムを組み立てていって，オリジナルな通信ソフトを作り上げていくのはさほど困難なことではないと思う．たとえば，アセンブラで書いた10KBくらいの超ミニ通信ソフト（今回の通信ライブラリのオブジェクトは3KB弱である）であるとか，個人的趣味に徹した私家版ソフトなど，個性的な通信ソフトの登場を心待ちにしている今日このごろである．通信ソフトはそう難しくない……本章を読んでそう思っていただけたらこれにまさる喜びはない．

パソコン通信用語集
Y-MODEM　X-MODEMの欠点を補う形で開発された転送手順．1ブロックは1024バイト単位で，最初にファイル名やタイムスタンプなどファイル名パケットを送る．

市販通信ライブラリ紹介

connect:232C
標準・拡張RS-232Cポート制御用C言語ライブラリ

connect:232Cは，Lattice C，MS-Cに対応した高機能なRS-232C制御用C言語ライブラリである．μPD8251 SIO調歩同期式全二重の動作モードの設定，ステータスの読み込み，受信バッファの変更などを簡単に行うことができ，RS-232Cを通してのデータアクセスをファイル操作と同様なオープン，クローズ，リード，ライトで行える．

また，通信速度は19,200bps，フロー制御はRTS/CTS信号によるハードウェアフロー制御をサポートしているため，MNPモデムにも対応可能である．

connect:232Cにはライブラリの完全なソースコードとサンプルとしてxmodemやymodemにも対応した通信ソフトが付属している．再販ライセンス付きなので，自分で作成したソフトを自由に販売・配布することができる．

製品名：connect:232C
対応機種：NEC PC-9801シリーズ
対応コンパイラ：Lattice C/DOS Ver.4.1，MS-C Ver.5.1/6.0
価格：38,000円
問い合わせ先：（株）ライフボート
TEL：03-3293-4711

RS-232C Library
プロフェッショナルな通信ライブラリ

RS-232C Libraryは，C言語でRS-232Cの各種制御を行うためのC言語ライブラリである．一般的な調歩同期式だけではなく，同期式についても完全にサポートしている．このため，BSCなどの同期式手順を使うアプリケーションにも対応可能である．

これら以外にも，半／全二重制御，ハード／ソフトフロー制御サポート，拡張RS-232Cボード（PC-9861K）対応などの特徴を備えている．

また，サンプルとしてY-TERMという通信ソフトが付属している．ライブラリ，サンプルのソースは完全に公開されており，組み込みローヤリティは無料である．

製品名：RS-232C Library
対応機種：NEC PC-9801シリーズ
対応コンパイラ：MS-C Ver.5.1以上，QuickC Ver.2.0以上
価格：23,000円
問い合わせ先：吉正電子（株）
TEL：03-5371-3041

RS-232Cが簡単
あの「簡単シリーズ」の通信ライブラリ

RS-232Cが簡単は，C言語用プログラミング支援ツールとして知られる「簡単シリーズ」の1つである．RS-232Cが簡単は，アプリケーションプログラムの通信用ドライバとして常駐し，非同期方式で通信を行う．

割り込み機能の設定を変更したり，ドライバを介さずに通信を行うことも可能である．また，割り込み処理の禁止，許可などを行うこともできる．サポートしている通信速度は，75bps～19,200bpsまでである．

パッケージにはサンプルプログラムが付属し，組み込みローヤリティは無料である．

製品名：RS-232Cが簡単
対応機種：NEC PC-9801（XA/XL/XL²/RLハイレゾ/LTを除く）
対応コンパイラ：MS-C，Turbo C，Quick C，Lattice C（使用するコンパイラを指定する）
価格：15,000円
問い合わせ先：エフエーシステムエンジニアリング（株）
TEL：0899-31-2886

C COMMUNICATIONS TOOLKIT
コミュニケーションC言語ライブラリ

C COMMUNICATIONS TOOLKITは，ヘイズ製またはそのコンパチブル，テレビット製，UDS製の各モデムおよびASCII，XMODEM，XMODEM/CRC，Kermit，YMODEM，YMODEM-Gなどのプロトコルに対応した通信用ライブラリである．

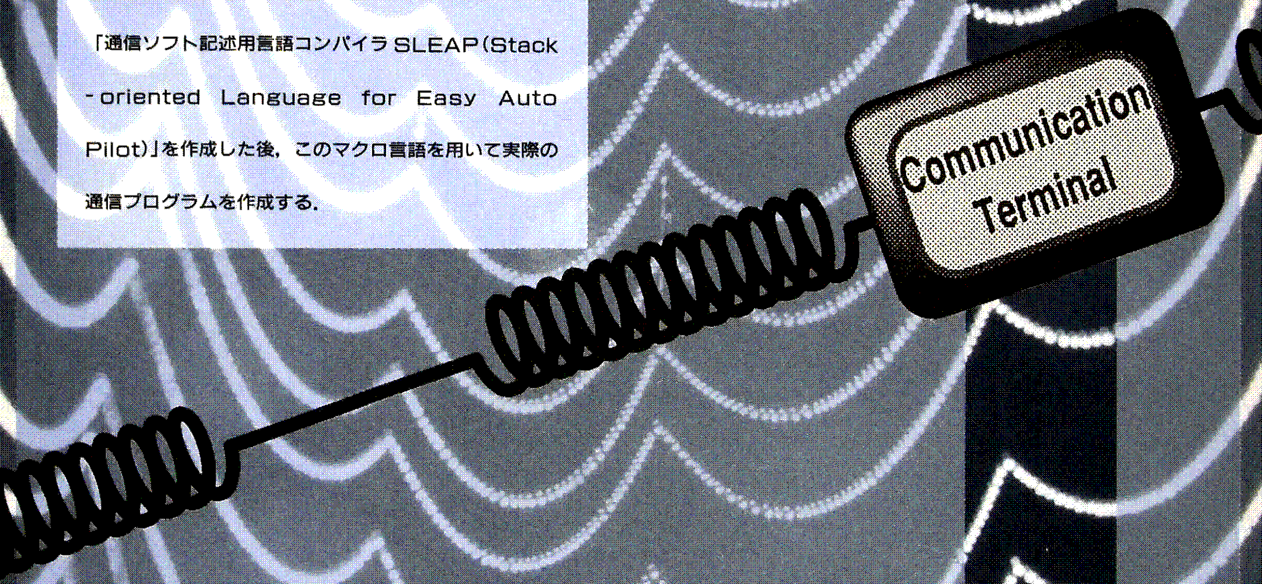
ハードウェア制御モジュールが独立設計されているため，多種のシリアルチップに対応することが可能で，通信ポート数には制限がない．通信スピードは，IBM PC/AT，PS/2の場合115,200bpsまで可能である．

ANSI規格の端末エミュレーションなどの豊富なサンプルプログラムとソースコードが付属している．組み込みローヤリティは無料で，マニュアルは英文となっている．

製品名：C COMMUNICATIONS TOOLKIT
対応機種：IBM PC/AT，PS/2
対応コンパイラ：MS-C Ver.5.1/6.0，Turbo C Ver.2.0，QuickC，Power C Ver.1.2以上，Watcom C
価格：29,000円
問い合わせ先：マイクロデータUSソフトウェア販売部
TEL：03-3232-3210

パソコン通信用語集
アーカイブ　本来は大型コンピュータでデータのバックアップをすること．バックアップの際，データを圧縮することからパソコンではおもにファイルの圧縮を指す．

通信プログラムを利用してネットワークなどにアクセスしているとホストプログラムの仕様変更などによって，自分の通信プログラムを変更しなければならなくなることが往々にしてある．そのたびにプログラムソースを直し，コンパイルし直すのではめんどうでしかたがない．そこで，機能の追加や変更をできるだけ簡単に実現するために「通信機能記述言語」という方法を用いる．言語の仕様としては，コンパイラとしての側面に加えて対話環境としての側面をもつFORTHライクなものとする．「通信ソフト記述用言語コンパイラSLEAP(Stack-oriented Language for Easy Auto Pilot)」を作成した後，このマクロ言語を用いて実際の通信プログラムを作成する．



特集・通信ソフトの作り方

第3章 実践編

超お手軽ターミナル

FORTH系通信言語SLEAPで通信を実現する

株式会社ジャステックラボ　加賀島英渡＋山本博昭

さて，やっと私の出番になった．いちばんおもしろい導入部とハイライトの通信ライブラリを取られてしまったので，こちらはシコシコと通信ソフトでも作ることにしよう．特集のテーマは「通信ソフトの作り方」なのだから，本来はこちらがメインテーマになるはずなのだが，通信ソフトでいちばん重要なものは，実は実際にRS-232Cとやり取りをする部分．これがライブラリ化されていて，しかもちゃんと動いてくれていれば，通信ソフトのほうは楽なものである．

基本部分は簡単なのだが……

第1章でもふれたが，通信ソフトといってもいろいろある．対象をパソコンにかぎったとしても広義の「通信ソフトウェア」ということでいえば，パソコン通信に使う市販の高性能なソフトや，ときにそれよりもさらに高性能なフリーソフトウェアやシェアウェアももちろんこのジャンルだし，パソコン同士をケーブルで結んでファイルの転送をするためのものも，「通信を行うソフト」という意味でこの範疇に入ってくるだろう．パソコンではないが，銀行のキャッシャサービスの端末機とどこか秘密理に設置されている大きなコンピュータをつないでいるものも，通信ソフトと呼んで差しつかえあるまい．

もちろんここで（個人的な興味は別として）銀行の大きなコンピュータと通信するなんてお話をしてもはじまらないので，一般にパソコンにおいて「通信ソフト」と呼ばれているところのものを対象に話を進めることにしよう．

この種の通信ソフトの基本部分は，キー入力があったらそれを送信し，受信データがあればそれを表示するというものである．したがって，いちばん基本的な通信ソフトをCで書いてみると，リスト1のようになる．

簡単すぎて情けないくらいだが，嘘ではない．これに初期化関数の呼び出しを付け加えて，今回紹介した通信ライブラリとリンクすれば，ほんとうにちゃんとした通信ソフトになるのだ．もちろんこのプログラムでネットワークにアクセスしようと思ったら，モデムの初期化コマンドの送信からホストへ電話をかけさせたりすることまでのすべてを自分の手で行わなければならないので，こういうものが「ちゃんとしている」かどうかは議論の分かれるところである．

最近の通信ソフトは，バックグラウンドでデータを受信しながらこれから送信するテキストを編集するなどという，とんでもない機能があたりまえになってしまった．楽しみに1つ自分でも作ってみようかなどとシコシコやっていると，外野から「あれ，エディタついてないのぉ，これぇ」などといわれてクサるのがオチといういやぁなご時勢である．

いいじゃないの，通信ソフトなんだから通信ができればさ．と，開き直っていちばん簡単なところから自分の手で1つ1つ作り上げていくのもまた楽しいものである．なんといっても自分の手で作った通信ソフトが実際にネットワークとつながってやり取りをはじめた瞬間というのは，いいトシしたおじさんでも「やったやった」と躍り上がってむこう三軒両隣から大顰蹙をかうほどの感激があるものだ．

通信ライブラリを組み上げて通信言語を作る

前章で紹介した通信ライブラリを使って，通信機能をもつプログラムを書いてみるということになった．作るうえで考えたのは，通信ソフトというものは使っていて結構ちょこちょこと機能を追加したり変更したくなったりする誘惑にかられやすいものだということである．また，パソコン通信を普段利用している方ならわかると思うが，ネットワークのホストプログラムの仕様変更などによって，こちらのプログラムの動作を合わせなくてはならないことが往々にしてある．そのたびにプログラムソースを直し，コンパイルしなおしてテストというのはなかなかたいへんである．

機能の追加や変更をできるだけ簡単にやるようなプログラムにするにはどうしたらよいか？　そこで思い当たったのは「通信機能記述言語」という方法である．「通信言語」といえば，フリーソフトウェアで「CUE」（DUDE氏作）というすばらしい作品がある（筆者も愛用させていただいている）．「CUE」の域まで到達するためにはとても長い期間と多大な手間がかかるが，まずは土台となる言語仕様を作り上げて，そこから少しずつ機能を充実させていこう．最初におおもとになる文法仕様を決め，最低限必要な単語をそろえていく．C言語で書くのはここまでである．実際の通信機能の記述は，こ

パソコン通信用語集
アボートタイマ　回線接続中に一定の時間，データの送受信がないときに回線を切断する機能のこと．

の通信言語コンパイラ用に決めた文法にしたがって「マクロプログラム」として書いていくことになる.

つまり，プログラムの機能追加や変更はすべてこの「マクロプログラム」を修正するだけで可能であり,「コンパイラ」自体はよほどのことがない限り手を入れる必要がないことになろう.

今回の通信言語の基本思想

というわけで，いわゆる1つの言語を設計することになった. 張り切っていままでにないユニークなものをと考えてみたのだが，だいたい筆者が考えつくようなものはいままでにだれかが発表している. 単に「変わった言語」というのならばできないことはないのだが，変わっているだけが取り柄というのでは，使いにくくてどうにもならない. しかたがないので，実績がある既存の言語をベースにして，マクロとしての使いやすさに重点を置いたものを設計することにした.

基本思想などと大仰な見出しを付けてしまったが，今回は言語自体の作成ではなく通信ソフトを作ることが主眼なので，あまり複雑な言語をこね回すことは避けたい. 構造が単純でいて言語として十分実用的なもので，しかも言語自体が対話環境になっているためデバッグが容易であるという理由でFORTHを選んだ.

FORTHという言語については去年の連載「MS-DOSツール・ラボ」で1度解説しているが，あのときに作った「FAST」というコンパイラは，FORTH言語のコンパイラとしての一面を取り出して開発用言語として特化したものであった. あのときのコラムの中で少し触れているが，FORTHという言語にはコンパイラとしての側面に加えて，対話環境としての側面がある. 今回設計した言語の紹介に先立ち，いま一度FORTHという言語を解説し，とくにこの対話機能を強調しておきたい.

対話機能とは

FORTHという言語はデータの受け渡しにスタックを使用する. Cなどでも関数にデータを渡すときにはスタックを経由するが，FORTHでは戻り値もまたスタックに積んで返すので，引数はもちろん戻り値も複数にすることができる.

たとえば，Cでカーソル位置(X, Y)を返す関数は変数のアドレスを引数として渡してそこにデータを入れてもらうようにするのが一般的だが，FORTHではX, Yをそのままスタックに積んで返すことができる. このような機構を実現するため，FORTHはCALL/RETURNなどの制御に使用するためのスタックとデータの受け渡し用スタックという2本だての管理を行っている.

すべてのデータがスタックを経由して受け渡されるため，あるロジックを呼び出す際に明示的な引数の記述をしない. 引数はそれが呼び出されたときにスタックに積まれているものだからである. したがって，FORTHのプログラムは単に処理単語の名前を並べていく形式になる.

リスト2は,

"MESSAGE"(を)"PRINT"(し)"CONNECT"(して)"ACCESS"(し)"DISCONNECT"(する)

というFORTHのプログラムである. わかりやすいようにカッコで送り仮名を付けてある. "PRINT" は表示すべきデータを必要とするが，これはその前の "MESSAGE" という処理がスタックに積んでいる.

また，FORTHにはCやPascalなどでいうところの「演算子」という概念がない. だからといって，加減乗除などの演算ができないわけではなく，事実FORTHにも「+」という記号がある. 記号といってしまったが，実はFORTHの「+」は普通の言語でいう演算子という特別なものではなく,「スタック上の2つの数値を加えて結果をスタックに返す」という処理なのである

LIST

リスト1

```
int     main()
{
        for(;;)
        {
                if(kbhit())             RS_send_byte(getch());
                if(RS_reccount())       putch(RS_rec_byte());
        }
}
```

リスト2

```
MESSAGE PRINT CONNECT ACCESS DISCONNECT
```

る．さらに数値を記述するというコーディングもまた「数値をスタックに積む」処理として解釈される．

たとえば「1000」と記述したら，これは「1000（という数値をスタックに積む）」処理として扱われるのである．したがって，リスト3のFORTHプログラムを実行するとスタックには「3000」という数値が積まれていることになる．

さきほどの例にならって送り仮名を補うと，

"1000"(と)"2000"(を)"+"(する)

ということになる．

FORTHのプログラミングとはこのように，必要な処理を順に並べていくことである．

さて，いまの2つの例を見て，読者はほかの言語とくらべて非常に異質のものを感じられたのではないだろうか．もちろんデータがスタックを経由して受け渡されることも異質といえば異質ではあるが，それだけではなく，もっと本質的な違いに気がつかれたに違いない．1つ1つの単語を（数値でさえも），「処理」として解釈して実行していくため，その単語をすぐに実行することができるのだ．次の例を見ていただきたい．

例：10と30を加算した結果を，「ans」というエリアに格納する．

Cで書くと

```
ans = 10 + 30 ;
```

FORTHで書くと

```
10 30 + ans !
```

「"10"(と)"30"(を)"+"(して)"ans"(というアドレスに)"!"(格納する)」という記述である．Cで書いたも

LIST

リスト3

```
1000 2000 +
```

リスト4

```
[ sample_word                       ; "sample_wordが単語名
      "Hellow " sprint              ; "Hellow "を文字列表示
      10 do                         ; 「loop」までを10回繰り返す
            'A' cprint              ; 'A'を一文字表示
            'b' cprint              ; 'b'を一文字表示
      loop                          ; ループ終端
]                                   ; 単語定義の終わり
```

のは，左から読んでいっても行の最後まで読まないと何の行動を起こすこともできないが，FORTHで書くと「10」「30」「+」……と読んでいった時点で，それぞれの処理ができることがおわかりいただけるだろうか．

この特性をいかして，FORTH系の言語は「対話機能」が実現できる．キー入力した単語をその場ですぐにコンパイルして実行することができるということで，どちらかというとこれはBASICに近い感覚かもしれない．通信ソフトを自作するうえで，この特徴は大きな武器になる．テストをしていて，うまくいかないときや新しい機能を追加したときなど，その場で修正してすぐに実行してみることができるのだ．いったん接続を切って，プログラムを修正・コンパイルしてまたつなぐという手間を繰り返すことを考えてみれば，このへんの利点はおわかりいただけるだろう．

FORTH系通信言語SLEAP

話が少々わき道へそれてしまったようである．今回の眼目である「通信ソフト」を作るのは，いままで説明したコンパイラでコンパイルする「マクロプログラム」で，ということになる．作っていく手順を簡単にまとめるとこんな感じになる．

①コンパイラ本体の作成．
②マクロプログラムのための基本単語を作る．
③通信用の汎用単語集を作る．
④汎用単語を応用して，通信マクロプログラムを組む．
⑤マクロプログラムをコンパイルして実行する．

通信ソフトの基本は先に書いたように実に単純なものだ．しかしマクロプログラムのコンパイラとなると，いやしくも言語処理系である．通信機能のほかに言語処理系としての機能をもたなくてはならないのは当然である．分岐構文や繰り返し，算術や論理演算用の単語も用意しなくてはならないだろう．

ここで発想の転換をしてみよう．通信ソフトに言語機能を付けるのではなく，ある程度汎用に設計した言語の命令群に通信の基本機能を付けてみてはどうか．通信ソフトが書きやすいような言語という仕立て方をするのである．どうせやるのなら実用的なものをということで，ネットワークにアクセス

パソコン通信用語集
エンコード　データを変換すること．ここではトランスペアレントでない通信条件の中でバイナリファイルを送るためにテキストファイルに変換すること．

るときの自動運転プログラムまでも簡単に実現できると公約してしまおう．

このような考えから，この「通信ソフト記述用言語コンパイラ」をSLEAP (Stack-oriented Language for Easy Auto Pilot) と名づけることにした．少々気恥ずかしいくらいな名前だが，筆者の心意気と思ってお許しいただきたい．

SLEAPの単語記述ルール

SLEAPによるマクロプログラムは，「単語」を順次実行していく．ある基本的な機能をもつ単語を組み合わせた集合に単語名をつけて，またその単語を使ってより大きな機能をもつ単語を作って……という手順である．単語の定義は“[単語名”で始まり，“]”で終わる．また，未定義単語への前方参照は許されない．つまり，まだ出てきていない単語を呼び出すことはできないので，必然的にプログラムのメイン単語が最後に定義されることになる．

では具体的にどう記述するかサンプルを書いてみよう（リスト4）．

基本的には，上の行から下の行へと実行されていくが，後述する分岐や繰り返しの構文もある．上記の例では，回数指定による繰り返し構文を使用している．

図3

```
<回数> do
        繰り返す処理
loop
```

以下に単語を記述していく上でのいくつかのルールをまとめる．

①記述する単語は，それぞれ1個以上の空白・TAB・改行のいずれかで区切る．

②単語名には字数・字種ともに事実上制限はない．ただし，空白やTABは中に含めることができない．また，数字のみの場合は数値とみなされる．

③数値は直接記述してよい．16進表記の場合は，先頭に「$」をつける．

④1文字をあらわすときは「'」で括る．

⑤文字列をあらわすときは「"」で括る．次の文字列は制御コードとして認識する．

　¥r：CRコード（0x0D）

　¥n：LFコード（0x0A）

⑥「；」が出てきた時点で，その行の右端までは無視する（コメント扱い）．

SLEAPの基本構文

プログラム言語であるので，当然条件分岐や繰り返し構文が必要になってくる．SLEAPは，以下の繰り返し構文をもっている．

・条件分岐

・回数指定による繰り返し

・条件判断による繰り返し

・無条件繰り返し

以下に各構文の記述方法を簡単に説明する．

【条件分岐】

「if～else～then」という構文である（図1）．else～の部分はなくてもかまわないが，「then」の記述は必須である．真／偽いずれかで行う処理は，複数行にわたってもよい．

図1

```
<条件>
if
        <条件>が真だった場合に行う処理
else
        <条件>が偽だった場合に行う処理
then
```

条件分岐には，もう1つこれと逆の構文も用意した．

「-if～else～then」という構文は，<条件>が「○○でなかったら」という書き方のほうがロジックが書きやすい場合に使用する（図2）．

図2

```
<条件>
-if
        <条件>が偽だった場合に行う処理
else
        <条件>が真だった場合に行う処理
then
```

図4

```
begin
        繰り返す処理
<条件> until
```

図5

```
begin
        繰り返す処理
repeat
```

表1　SLEAPの基本単語一覧

●構文制御ワード	
if	スタックトップの値が真であればthenまたはelseまでを実行する
-if	スタックトップの値が偽であればthenまたはelseまでを実行する
else	ifまたは-ifの逆の状態のときにthenまでを実行する
then	if(-if)～の終端
do	スタックトップの数値分だけloopまでを実行する
loop	do～の終端
begin	until/repeatの始端
until	スタックトップの値が真であればbeginの次から実行を繰り返す
repeat	beginの次から実行を繰り返す
dup	スタックトップを複製してスタックに積む
?dup	スタックトップが真ならば複製する
drop	スタックトップを捨てる
swap	スタックトップと2番目の内容を入れ替える
over	スタックの2番目の内容を複製してスタックに積む
pick	任意のスタック内容を複製してスタックに積む
cprint	スタックトップの内容を1文字表示
sprint	スタックトップの内容を文字列表示
bprint	スタックトップの内容を1バイト16進表示
wprint	スタックトップの内容を1ワード16進表示
nprint	スタックトップの内容を数値表示
?cinput	キー入力検査して結果をスタックに積む
cinput	1文字入力して結果をスタックに積む
sinput	文字列入力して結果をスタックに積む
slen	文字列長を得て結果をスタックに積む
exit	スタックトップをリターンコードとしてDOSに戻る
exitn	0をリターンコードとしてDOSに戻る
return	呼び出し側に戻る（do～loopの中では使用不可）
]	単語定義の終端
+	スタックトップとスタックの2番目を加えて結果をスタックに積む
-	スタックトップからスタックの2番目を引いて結果をスタックに積む
*	スタックトップとスタックの2番目を乗じて結果をスタックに積む
/	スタックトップをスタックの2番目で割って結果をスタックに積む
and	スタックトップとスタックの2番目のビットごとのAND演算を行って結果をスタックに積む
or	スタックトップとスタックの2番目のビットごとのOR演算を行って結果をスタックに積む
xor	スタックトップとスタックの2番目のビットごとのXOR演算を行って結果をスタックに積む
not	スタックトップの論理否定を行って結果をスタックに積む
cmp	スタックトップの1の補数をスタックに積む
shl	スタックトップの指定ビット分でスタックの2番目のデータを左シフトして結果をスタックに積む
shr	スタックトップの指定ビット分でスタックの2番目のデータを右シフトして結果をスタックに積む
1+	スタックトップの値をインクリメント
1-	スタックトップの値をデクリメント
=	スタックトップとスタックの2番目が等しいか？の結果をスタックに積む
>	スタックの2番目がスタックトップより大きいか？の結果をスタックに積む
<	スタックの2番目がスタックトップより小さいか？の結果をスタックに積む

パソコン通信用語集
リライアブルバッファ機能　エラー訂正機能のないモデムをオートリライアブルで接続するとき，一定数のキャラクタを受信した場合はエラー訂正機能なしと判断させる機能

@	スタックトップの指定アドレスからダブルワードのメモリロードを行ってスタックに積む
w@	スタックトップの指定アドレスからワードのメモリロードを行ってスタックに積む
b@	スタックトップの指定アドレスからバイトのメモリロードを行ってスタックに積む
!	スタックトップの指定アドレスへスタックの2番目のダブルワードのメモリストア
w!	スタックトップの指定アドレスへスタックの2番目のワードのメモリストア
b!	スタックトップの指定アドレスへスタックの2番目のバイトのメモリストア
1+!	スタックトップの指定アドレスのダブルワードのインクリメント
1-!	スタックトップの指定アドレスのダブルワードのデクリメント
+!	スタックトップの指定アドレスのダブルワードのスタックトップ分増加
-!	スタックトップの指定アドレスのダブルワードのスタックトップ分減少
s!	スタックトップの指定アドレスにスタックの2番目の文字列をコピーする
s+	スタックトップの指定アドレスにスタックの2番目の文字列を追加する
scomp	スタックトップとスタックの2番目の2つの文字列を大小関係で比較して結果をスタックに積む
s=	スタックトップとスタックの2番目の2つの文字列の一致結果をスタックに積む
fopen	スタックトップのファイル名文字列のファイルをオープンする．FILE構造体へのポインタをスタックに積む
fclose	スタックトップのファイルをクローズする．成功したら0を，失敗したら0以外の値をスタックに積む
fcprint	スタックトップのファイルにスタックの2番目の1文字を書き込む
fsprint	スタックトップのファイルにスタックの2番目の文字列を書き込む
fcinput	スタックトップのファイルから1文字を読み込んでスタックに積む
fsinput	スタックトップのファイルから文字列を読み込んでスタックに積む
system	スタックトップのコマンド文字列をDOSコマンド実行
systemwords	分岐構文ワードリストの一覧表示
userwords	ユーザー定義ワードの一覧表示
definewords	システム定義ワードの一覧表示
help	上記3ワードを一度に実行する
●RS-232Cライブラリ呼び出し用ワード	
rs_initialize	RS-232C初期化
rs_finish	通信使用終了
rs_send_data	スタックトップのデータを送信バッファに1バイト書き込む
rs_receive_data	受信バッファから1バイト取り出してスタックに積む
rs_peek_data	受信バッファから1バイト読み出してスタックに積む
rs_send_buffer_space	送信バッファ余裕バイト数取得してスタックに積む
rs_receive_count	受信済みデータ数取得してスタックに積む
rs_send_count	送信待ちデータ数取得してスタックに積む
rs_xcontrol_on	Xコントロールモード ON
rs_xcontrol_off	Xコントロールモード OFF
rs_rts_on	RTS（RS）　ON
rs_rts_off	RTS（RS）　OFF
rs_dtr_on	DTR（ER）　ON
rs_dtr_off	DTR（ER）　OFF
rs_break_on	ブレーク信号　ON
rs_break_off	ブレーク信号　OFF
rs_xcontrol_check	Xコントロールモード検査結果をスタックに積む
rs_dcd_check	DCD（CD）検査結果をスタックに積む
rs_cts_check	CTS（CS）検査結果をスタックに積む
rs_dsr_check	DSR（DR）検査結果をスタックに積む
rs_set_timer	スタックトップのタイムアウト時間でタイムアウト監視用タイマ設定
rs_chk_timer	タイムアウトチェックした結果をスタックに積む
●RS-232Cライブラリ初期設定用ワード	
rs_set_baud_rate	スタックトップの値でボーレート設定
rs_set_data_length	スタックトップの値でキャラクタ長設定
rs_set_parity_bit	スタックトップの値でパリティ種別設定
rs_set_stop_bit	スタックトップの値でストップビット長設定
rs_set_xcontrol	スタックトップの値でXコントロール設定

パソコン通信用語集
オーバーフロー　端末やモデムのバッファがデータでいっぱいになり，残りのデータが消失すること．フロー制御が正しく行われていないと起こる．

【回数指定による繰り返し】

「do」が出てきた時点でスタックに積まれている数値の回数だけ「loop」までの処理を繰り返す．繰り返す処理は複数行にわたって記述してもよい．あらかじめ繰り返す回数がわかっている場合に使用する（図3）．

【条件判断による繰り返し】

「until」が出てきた時点でスタックに積まれている条件が真であれば，「begin」の次から実行を繰り返す．偽であれば，繰り返しを終了して「until」の次を実行する．繰り返す処理は複数行にわたって記述してもよい（図4）．

【無条件繰り返し】

「repeat」が出てくると，無条件に「begin」の次から実行を繰り返す（図5）．繰り返す処理は複数行にわたって記述してもよい．

ただし，これだけでは永遠に終らないプログラムになってしまうので，なんらかの脱出条件が必要である．典型的なのは，条件分岐と「return」という単語を組み合わせて脱出する方法である．「return」は，実行中の単語を呼び出した単語へ戻るという機能である．繰り返しの脱出条件が複雑なときや，脱出すべき箇所が複数ある場合には「until」よりもこちらを使うことになろう．

SLEAPのもつ制御構文は以上である．これらの制御構文を使って，単語を組み合わせて通信プログラムを作っていく．とはいっても，一から作り出すのでは通常のコンパイラで作るのとさして手間が変わらないことになってしまうため，あらかじめ基本的な機能をもつ単語を内蔵することにする．通信言語ならではのRS-232C関係の制御用単語も用意する．まずはこの基本単語の一覧をご覧いただきたい（表1）．

注：説明中，「スタックトップ」，「スタックの2番目」といった用語が出てくるが，「スタックトップ」というのはその時点で最後にスタックに積まれたデータをさす．同様に「スタックの2番目」というのは，スタックトップの1つ前に積まれたデータである．

SLEAPの基本単語一覧

一覧だけではものたりないという方は，ぜひ今後出てくるマクロプログラムの実例集を参考にしていただきたい．また，通信ライブラリの呼び出し単語は，ライブラリの関数と1対1に対応しているので，機能面はライブラリの仕様書がそのままあてはまる．

SLEAPの使い方

誰が何といおうと，SLEAPはコンパイラである．ただし，コンパイラと

リスト5 SLEAP.C

```
/* =====================================================================

    SLEAP.C          Copyright 1991 bY JUSTECH-LABO. (H.kanoshima)

===================================================================== */

#include    <dos.h>
#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <jstring.h>
#include    <ctype.h>
#include    <jctype.h>
#include    <conio.h>
#include    <setjmp.h>

#include    "forthcom.h"
#include    "compile.h"

#define     ENVSOURCE   "SLEAP"

#define     SDATA_MAX   32

struct sdt{
    char    *sname;
    FILE    *sfile;
    int      sline;
}sdata[SDATA_MAX];                          /* ソースファイルリスト */

struct  sdt *sdtptr = &sdata[SDATA_MAX];

char        *sourcename;                    /* コンパイル中ソースファイル名 */
FILE        *sourcefile;                    /* コンパイル中ソースファイル */
int         linecount;                      /* ソース行数 */

char        sourceline[SRCLINE_MAX];        /* 処理中ソーステキスト */
char        *sourceptr;                     /* ↑用ポインタ */

BOOL        directin = BFALSE;              /* キーボードからの入力でTRUEになる */
                                            /* いったんTRUEになったら、最後まで */
                                            /* FALSE にはならない。 */

BOOL        direct_compile = BFALSE;        /* キーボードからの"compile" 実行。 */

/* =====================================================================
キーボードからの直接入力の場合にBTRUEを返す。
===================================================================== */

BOOL    is_directin(void)
{
    return  directin;
}

/* =====================================================================
処理中のソースファイル名、行数を返す関数とそれらを表示する関数。
===================================================================== */

static  char    *get_sourcename(void)
{
    return  sourcename;
}

static  int     get_sourceline(void)
{
    return  linecount;
}

void    print_source_and_line(void)
{
    if(is_directin() == BTRUE)  return;
    printf("'%s' - %04d ",get_sourcename() , get_sourceline());
}

/* =====================================================================
つぎのソースファイルを準備する。指定されたソースファイルを全部読み終えた場
合（最初からひとつも指定されていなかった場合も含む）は、標準入力から読み込
```


パソコン通信用語集
オンラインモード　データモードともいう．モデムはダイヤル後，接続完了から回線切断までこのモードになる．モデムは端末から入力された文字をすべて回線へ出力する．

はいってもMS-DOSで実行できる「COM」とか「EXE」ファイルを出力するわけではない．SLEAPがコンパイルするのは，前節で一覧にした単語を使って記述したマクロプログラムであり，それを解析して中間コードを出力する．

また，SLEAPの特徴の1つとして，「即実行モード」がある．BASICをご存じの方なら，すぐにピンとくるかもしれないが，SLEAPの単語は「単語定義」を使ってプログラムを組むだけではなく，その単語をSLEAP上で直接入力すると，ただちに実行できるのだ．使い方の例をあげてみよう(図6)．

「'a'」という文字を画面表示させるためには，図6の例のように「abc」などという単語を定義して実行してもよいし，図7のように直接cprintを実行させることも可能なのである．

SLEAPのためのプログラムは，ファイルにすることもできる．ファイル名は任意でよいが，ほかのファイルと

図6

```
>>[ abc 'a' cprint ]
>>abc
a
```

図7

```
>>'a' cprint
a
```

```
A>SLEAP
***** SLEAP ver 1.0 Copyright Justech labo. *****

>> compile stdlib.s
Compiled : 32DE - 32F5 (crlf)
Compiled : 32F6 - 3302 (npr)
Compiled : 3303 - 330F (wpr)
Compiled : 3310 - 331D (@npr)
Compiled : 331E - 332C (2drop)
Compiled : 332D - 333B (3drop)
Compiled -stdlib.s-
Stack dump :
>>
```

図8 SLEAP内でコンパイルする方法

図9

```
A>SLEAP STDLIB.S
***** SLEAP ver 1.0 Copyright Justech labo. *****

Compiled -stdlib.s-
>>
```

```
むようになる。
======================================================================= */

static  void    set_nextsource(void)
{
    if(sourcefile != (FILE *)NULL)  fclose(sourcefile);

    if(sdtptr == &sdata[SDATA_MAX])     exit(0);

    if(sdtptr->sfile == (FILE *)NULL)
    {
        sdtptr->sfile = sourcefile = fopen(sdtptr->sname , "r");
        if(sourcefile == (FILE *)NULL)
        {
            printf("Can't open '%s'¥n" , sdtptr->sname);
            exit_error(ERROR_NOFILE);
        }
        linecount = 0;
    }
    else
    {
        sourcefile = sdtptr->sfile;
        linecount  = sdtptr->sline;
    }
    sourcename = sdtptr->sname;
    sdtptr++;

    if(isatty(fileno(sourcefile)))
    {
        directin = BTRUE;
        direct_compile = BFALSE;
    }
}

/* =======================================================================
ソースファイルを登録する。
======================================================================= */

static  void    set_newsource(char *sptr , FILE *fptr)
{
    if(sdtptr == sdata)
    {
        printf("ソースファイルが多すぎます(%d個)",SDATA MAX);
            exit(ERROR_SOURCEMAX);
    }
    sdtptr--;
    sdtptr->sfile = fptr;
    sdtptr->sname = sptr;
    sdtptr->sline = linecount;

    sourcefile = (FILE *)NULL;
}

/* =======================================================================
つぎのソース行を準備する。
======================================================================= */

static  void    set_nextline(void)
{
    for(;;)
    {
        if(isatty(fileno(sourcefile)))  printf(">> ");
        if(fgets(sourceline , SRCLINE_MAX , sourcefile) == NULL)
            set_nextsource();
        else
            break;
    }
    linecount++;
    sourceptr = sourceline;
}

/* =======================================================================
ソース行バッファを初期化する。（エラー時の対策）
======================================================================= */

void    init_sourceline(void)
{
    sourceptr = sourceline;
    *sourceptr = '¥0';
}

/* =======================================================================
ソース行に、指定された文字列を空白を一つ置いて追加する。
======================================================================= */

void    cat_sourceline(char *sptr)
```

パソコン通信用語集
シェアウェア　試用するためにフリーソフトウェア同様に無料で入手することができる．使い続けるには代金を支払い，必要がなければディスクから削除する必要がある．

```
{
    strcat(sourceline , " ");
    strcat(sourceline , sptr);
}

/* ======================================================================
ソース行から指定されたバッファに文字列を切り出す。"¥t","¥r","¥n","¥e" の各
エスケープシーケンスを処理する。
====================================================================== */

static  void    get_word_str(char *wptr , char stopchar)
{
    sourceptr++;
    do
    {
        if(*sourceptr == '¥¥')
        {
        char    escchar;

            sourceptr++;
            escchar = *sourceptr++;
            switch(escchar)
            {
                case    '¥0':   sourceptr--;        break;
                case    't':    *wptr++ = '¥t';     break;
                case    'r':    *wptr++ = '¥r';     break;
                case    'n':    *wptr++ = '¥n';     break;
                case    'e':    *wptr++ = 0x1b;     break;
                default:        *wptr++ = escchar;
                                if(iskanji(escchar))    *wptr++ = *sourceptr++;
            }
        }
        else    *wptr++ = *sourceptr++;
    }while((*sourceptr != '¥0') && (*sourceptr != stopchar));

    if(*sourceptr == stopchar)  sourceptr++;

    *wptr = '¥0';
}

/* ======================================================================
ソース行から文字定数を切り出す。
====================================================================== */

static  void    get_word_char(char *wptr)
{
    get_word_str(wptr , 0x27);
}

/* ======================================================================
ソース行から文字列定数を切り出す。
====================================================================== */

static  void    get_word_string(char *wptr)
{
    get_word_str(wptr , '"');
}

/* ======================================================================
ソース行から一般のワードを切り出す。数値も含まれる。この段階では数値である
かどうか、また未定義のワードであるかどうかの判定は行っていない。文字定数、
文字列定数以外のものすべてであると考えてよい。
====================================================================== */

static  void    get_word_word(char *wptr)
{
    while(*sourceptr > ' ') *wptr++ = *sourceptr++;
    *wptr = '¥0';
}

/* ======================================================================
ソース行から指定されたバッファに単語を切り出す。単語の区切りは半角の空白以
下の値を持つキャラクタコード('¥0'は除く)である。ソース行を最後まで処理し
終えたらつぎのソース行を自動的に読み込む。戻り値は単語先頭のキャラクタ。
====================================================================== */

static  char    get_word_sub(char *wptr)
{
char    topch;

    for(;;)
    {
        while((*sourceptr != '¥0') && (*sourceptr <= ' '))  sourceptr++;
        if((*sourceptr == '¥0') || (*sourceptr == ';'))     set_nextline();
        else    break;
    }
    topch = *sourceptr;
    switch(topch)
    {
        case    0x27:   get_word_char(wptr);
                        break;
        case    '"':    get_word_string(wptr);
                        break;
        default:        get_word_word(wptr);
    }
    return  topch;
}

/* ======================================================================
単語切り出しのメイン関数。指定されたバッファに切り出した単語をセットすると
ともに、戻り値として単語の属性を返す。この段階では一般ワードの単語属性は検
査していない。
====================================================================== */

static  BOOL    is_alldigit(char *sptr)
{
    while(*sptr != '¥0')
    {
        if(isdigit(*sptr) == 0) return  BFALSE;
        sptr++;
    }
    return  BTRUE;
}

static  BOOL    is_truenumber(char topch , char *sptr)
{
    if(isdigit(topch))  return  is_alldigit(sptr);
    if(topch != '$')    return  BFALSE;
    sptr++;
    while(*sptr != '¥0')
    {
        if(isxdigit(*sptr) == 0)    return  BFALSE;
        sptr++;
    }
    return  BTRUE;
}

/* ======================================================================
単語切り出しのメイン関数。指定されたバッファに切り出した単語をセットすると
ともに、戻り値として単語の属性を返す。この段階では一般ワードの単語属性は検
査していない。
====================================================================== */

IDCODE  get_word(char *wptr)
{
char    topch;

    topch = get_word_sub(wptr);

    if(strcmp(wptr , "[") == 0)             return  ID_LABEL;
    if(strcmp(wptr , ENDWORD) == 0)         return  ID_ENDLABEL;
    if(strcmp(wptr , "var") == 0)           return  ID_VAR;
    if(strcmp(wptr , "buffer") == 0)        return  ID_BUFFER;
    if(strcmp(wptr , "compile") == 0)       return  ID_COMPILE;
    if(strcmp(wptr , "end") == 0)           return  ID_END;
    if(topch == 0x27)                       return  ID_CHAR;
    if(topch == '"')                        return  ID_STRING;
    if(is_truenumber(topch, wptr) == BTRUE) return  ID_NUMBER;
    return  ID_WORD;
}

/* ======================================================================
追加コンパイルを行う。
====================================================================== */

static  void    compile_file(void)
{
static  char    wordbody[WORDLINE_MAX];

        (void)get_word(wordbody);

        if(is_directin() == BTRUE)  direct_compile = BTRUE;

        sdtptr--;
        set_newsource(wordbody , NULL);

        set_nextsource();
        set_nextline();
}

/* ======================================================================
システム全体のイニシャライズを行う。
====================================================================== */

static  void    set_envsource(char *sptr)
{
char    *nptr;

    nptr = strtok(sptr , " ,");
    if(nptr == NULL)    return;

    set_envsource(NULL);

    set_newsource(nptr , NULL);
}

static  void    set environ_source(void)
{
char    *envptr;

    envptr = getenv(ENVSOURCE);
    if(envptr == NULL)  return;

    envptr += strspn(envptr , " ,");
    if(*envptr == '¥0') return;

    set_envsource(envptr);
}

static  void    init_main(int argc, char **argv)
{
int     i;

    sourcefile = (FILE *)NULL;
    sourceptr = sourceline;
    *sourceline = '¥0';
    linecount = 0;

    set_newsource("(STDIN)" , stdin);

    for(i = argc - 1 ; i ; i--)
    {
        set_newsource(argv[i] , NULL);
    }

    set_environ_source();

    set_nextsource();
    set_nextline();

    if(buffer_init() == BFALSE)
    {
        printf("バッファが確保できません¥n");
        exit(ERROR_OBJFULL);
    }
}

/* ======================================================================
メイン関数である。
====================================================================== */

```


パソコン通信用語集

シリアルポート　モデムから見て端末と接続する側。RS-232Cインターフェースを指す。シリアルポートはモデムポートの論理伝送速度より速い伝送速度の必要がある。

```
extern      jmp_buf env;

int     main(int argc,char **argv)
{
static  char    wordbody[WORDLINE_MAX];
static  IDCODE  id;
int     jmpcode;

extern  void    rs_finish();

    printf("***** SLEAP ver 1.0 Copyright Justech labo. *****¥n¥n");

    atexit(rs_finish);

    init_main(argc , argv);

    jmpcode = setjmp(env);
    if(jmpcode != 0)
    {
        if(direct_compile == BTRUE)
        {
            if(jmpcode != DIRECT_NORMALEND)
            {
                while(direct_compile == BTRUE)
                {
                    set_nextsource();
                    /* ↑direct_compileはこの中でキーボードから */
                    /*   の入力に切り替わったときにBFALSEになる */
                    set_nextline();
                }
            }
        }
        if(is_directin() == BTRUE)  dump_dstack();
    }
    for(;;)
    {
        set_symp_and_objp();
        id = get_word(wordbody);
        switch(id)
        {
            case    ID_LABEL:   definition_word();
                                break;
            case    ID_VAR:     definition_var();
                                break;
            case    ID_BUFFER:  definition_buffer();
                                break;
            case    ID_END:     execute_word();
                                break;
            case    ID_COMPILE: compile_file();
                                break;
            default:            direct_execution(id , wordbody);
        }
        report();
    }
}
```

リスト6 SLEAP_H

```
BOOL    is_directin(void);
void    init_sourceline(void);
void    cat_sourceline(char *);
/* char *get_sourcename(void);
int     get_sourceline(void); */
void    print_source_and_line(void);
IDCODE  get_word(char *);
```

リスト7 INTERP_C

```
/* ==================================================================

    INTERP.C            Copyright 1991 bY JUSTECH-LABO. (H.kanoshima)

================================================================== */

#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <setjmp.h>

#include    "forthcom.h"
#include    "object.h"
#include    "sleap.h"
#include    "rs232lib.h"

#define     RSP_MAX         256     /* リターンスタックの要素数 */
#define     DSP_MAX         256     /* データスタックの要素数 */

/* ------------------------------------------------------------------
リターン/データ両スタックの定義。
------------------------------------------------------------------ */

DSTKV   dsp[DSP_MAX];           /* データスタック */
DSTKV   *dspp = &dsp[DSP_MAX];  /* インデクス */

RSTKV   rsp[RSP_MAX];           /* リターンスタック */
RSTKV   *rspp = &rsp[RSP_MAX];  /* インデクス */

/* ------------------------------------------------------------------
実行時のプログラムカウンタである。
------------------------------------------------------------------ */

OBJ     pcount;

/* ==================================================================
コンパイラが暗黙のうちに生成する特殊な関数。
================================================================== */

void    f_push_string(void)
{
    *(--dspp) = (DSTKV) pcount.cptr + sizeof(int);
    pcount.cptr += *pcount.iptr;
}

void    f_push_buffer(void)
{
    *(--dspp) = (DSTKV) pcount.cptr + sizeof(int);
    pcount.cptr += *pcount.iptr;
}

void    f_push_wnumber(void)
{
    *(--dspp) = (DSTKV) *pcount.iptr++;
}

void    f_push_lnumber(void)
{
    *(--dspp) = (DSTKV) *pcount.lptr++;
}

void    f_doentry(void)
{
    (--rspp)->stknum = *(dspp++);
    (--rspp)->stknum = 0;
}

void    f_loopentry(void)
{
DSTKV       i,j;

    i = rspp->stknum;
    j = (rspp + 1)->stknum;
    if(++i < j)
    {
        rspp->stknum = i;
        pcount.pptr = pcount.bptr->pcptr;
    }
    else
    {
        rspp += 2;
        pcount.bptr++;
    }
}

void    f_jmp(void)
{
    pcount.pptr = pcount.bptr->pcptr;
}

void    f_truejmp(void)
{
    if(*dspp++ != 0)
        pcount.pptr = pcount.bptr->pcptr;
    else
        pcount.bptr++;
}

void    f_falsejmp(void)
{
    if(*dspp++ == 0)
        pcount.pptr = pcount.bptr->pcptr;
    else
        pcount.bptr++;
}

/* ==================================================================
一般関数。ユーザが自由に使用できる。これを追加定義した場合は必ずテーブルに
登録しておくこと。
================================================================== */

#define f_pop()     (*(dspp++))
#define f_push(x)   (*(--dspp) = x)

/* ------------------------------------------------------------------ */

```

パソコン通信用語集

ストップビット　無手順伝送のとき1バイトの始まりを示すために先頭で1ビット0を送る。これがスタートビット。逆に1バイトの終りを示すためのビットのこと。

```
static  void    f_dup(void)
{
DSTKV   data;

    data = *dspp;
    f_push(data);
}

/* ----------------------------------------------------------------- */

static  void    f_truedup(void)
{
    if(*dspp)   f_dup();
}

/* ----------------------------------------------------------------- */

static  void    f_drop(void)
{
    dspp++;
}

/* ----------------------------------------------------------------- */

static  void    f_rdrop(void)
{
    rspp++;
}

/* ----------------------------------------------------------------- */

static  void    f_swap(void)
{
DSTKV   data;

    data = *dspp;
    *dspp = *(dspp + 1);
    *(dspp + 1) = data;
}

/* ----------------------------------------------------------------- */

static  void    f_rot(void)
{
DSTKV   data;

    data = *(dspp + 2);
    *(dspp + 2) = *(dspp + 1);
    *(dspp + 1) = *dspp;
    *dspp = data;
}

/* ----------------------------------------------------------------- */

static  void    f_rrot(void)
{
DSTKV   data;

    data = *dspp;
    *dspp = *(dspp + 1);
    *(dspp + 1) = *(dspp + 2);
    *(dspp + 2) = data;
}

/* ----------------------------------------------------------------- */

static  void    f_over(void)
{
DSTKV   data;

    data = *(dspp + 1);
    f_push(data);
}

/* ----------------------------------------------------------------- */

static  void    f_pick(void)
{
int     idx;
DSTKV   data;

    idx = f_pop();
    data = *(dspp + idx);
    f_push(data);
}

/* ----------------------------------------------------------------- */

static  void    f_cprint(void)
{
    putch(f_pop());
}

/* ----------------------------------------------------------------- */

static  void    f_sprint(void)
{
    printf("%s", (char *)f_pop());
}

/* ----------------------------------------------------------------- */

static  void    f_bprint(void)
{
    printf("%02X", f_pop());
}

/* ----------------------------------------------------------------- */

static  void    f_wprint(void)
{
    printf("%04X", f_pop());
}

/* ----------------------------------------------------------------- */

static  void    f_nprint(void)
{
    printf("%ld", f_pop());
}

/* ----------------------------------------------------------------- */

static  void    f_qcinput(void)
{
    f_push(kbhit());
}

/* ----------------------------------------------------------------- */

static  void    f_cinput(void)
{
    f_push(getch());
}

/* ----------------------------------------------------------------- */

static  void    f_sinput(void)
{
    gets((char *)f_pop());
}

/* ----------------------------------------------------------------- */

static  void    f_slength(void)
{
    *dspp = strlen((char *) *dspp);
}

/* ----------------------------------------------------------------- */

void    f_endword(void)
{
    pcount.pptr = (rspp++)->stkptr;
}

/* ----------------------------------------------------------------- */

static  void    f_exit(void)
{
    exit((int)f_pop());
}

/* ----------------------------------------------------------------- */

static  void    f_exitn(void)
{
    exit(0);
}

/* ----------------------------------------------------------------- */

static  void    f_rsptop(void)
{
    *(--dspp) = rspp->stknum;
}

/* ----------------------------------------------------------------- */

static  void    f_rsp2nd(void)
{
    *(--dspp) = (rspp + 1)->stknum;
}

/* ----------------------------------------------------------------- */

static  void    f_todsp(void)
{
    *(--dspp) = (rspp++)->stknum;
}

/* ----------------------------------------------------------------- */

static  void    f_torsp(void)
{
    (--rspp)->stknum = *(dspp++);
}

/* ----------------------------------------------------------------- */

static  void    f_leave(void)
{
    rspp->stkptr = (rspp + 1)->stkptr;
}

/* ----------------------------------------------------------------- */

static  void    f_return(void)
{
    pcount.pptr = (rspp++)->stkptr;
}

/* ----------------------------------------------------------------- */

static  void    f_add(void) { *(dspp + 1) += *dspp; dspp++; }
static  void    f_sub(void) { *(dspp + 1) -= *dspp; dspp++; }
static  void    f_mul(void) { *(dspp + 1) *= *dspp; dspp++; }
static  void    f_div(void)
{
    if(*dspp == 0)
    {
        printf("い、いかん、0で割ろうとしているではないかｯ!¥n");
        return;
    }
```


パソコン通信用語集

全2重　シリアルインターフェースの仕様で同時に双方向の伝送を行うことができる方式。交互に切り替えて伝送する方式を半2重という。

```c
    *(dspp + 1) /= *dspp; dspp++;
}

/* ----------------------------------------------------------------- */

static  void    f_and(void) { *(dspp + 1) &= *dspp; dspp++; }
static  void    f_or(void)  { *(dspp + 1) |= *dspp; dspp++; }
static  void    f_xor(void) { *(dspp + 1) ^= *dspp; dspp++; }

/* ----------------------------------------------------------------- */

static  void    f_not(void) { *dspp = !*dspp; dspp++; }
static  void    f_cmp(void) { *dspp = ~*dspp; dspp++; }

/* ----------------------------------------------------------------- */

static  void    f_shl(void) { *(dspp + 1) <<= *dspp; dspp++; }
static  void    f_shr(void) { *(dspp + 1) >>= *dspp; dspp++; }

/* ----------------------------------------------------------------- */

static  void    f_inc(void) { (*dspp)++; }
static  void    f_dec(void) { (*dspp)--; }

/* ----------------------------------------------------------------- */

static  void    f_equ(void)
    { *(dspp + 1 ) = *dspp == *(dspp + 1) ? 1 : 0; dspp++; }
static  void    f_big(void)
    { *(dspp + 1 ) = *(dspp + 1)  > *dspp ? 1 : 0; dspp++; }
static  void    f_sml(void)
    { *(dspp + 1 ) = *(dspp + 1)  < *dspp ? 1 : 0; dspp++; }

/* ----------------------------------------------------------------- */

static  void    f_load(void)    { *dspp = *(DSTKV *)*dspp; }
static  void    f_wload(void)   { *dspp = *(unsigned int *)*dspp; }
static  void    f_bload(void)   { *dspp = *(char *)*dspp; }

/* ----------------------------------------------------------------- */

static  void    f_store(void)
{
DSTKV   *lptr;

    lptr = (DSTKV *)f_pop();
    *lptr = f_pop();
}

static  void    f_wstore(void)
{
unsigned    int *iptr;

    iptr = (unsigned int *)f_pop();
    *iptr = f_pop();
}

static  void    f_bstore(void)
{
char    *cptr;

    cptr = (char *)f_pop();
    *cptr = f_pop();
}

static  void    f_incstore(void)
{
DSTKV   *lptr;

    lptr = (DSTKV *)f_pop();
    (*lptr)++;
}

static  void    f_decstore(void)
{
DSTKV   *lptr;

    lptr = (DSTKV *)f_pop();
    (*lptr)--;
}

static  void    f_addstore(void)
{
DSTKV   *lptr;

    lptr = (DSTKV *)f_pop();
    *lptr += f_pop();
}

static  void    f_substore(void)
{
DSTKV   *lptr;

    lptr = (DSTKV *)f_pop();
    *lptr -= f_pop();
}

/* ----------------------------------------------------------------- */

static  void    f_scopy(void)
{
    strcpy((char *)*(dspp + 1) , (char *)*dspp);
    dspp += 2;
}

static  void    f_scat(void)
{
    strcat((char *)*(dspp + 1) , (char *)*dspp);
    dspp += 2;
}

static  void    f_scmp(void)
{
    *(dspp + 1) = strcmp((char *)*(dspp + 1) , (char *)*dspp);
    dspp++;
}

static  void    f_sequ(void)
{
    *(dspp + 1) = strcmp((char *)*(dspp + 1) , (char *)*dspp) == 0 ? 1 : 0;
    dspp++;
}

/* ----------------------------------------------------------------- */

static  void    f_open(void)
{
    *(dspp + 1) = (DSTKV)fopen((char *)*(dspp + 1) , (char *)*dspp);
    dspp++;
}

static  void    f_close(void)
{
    *dspp = fclose((FILE *)*dspp);
}

static  void    f_wchr(void)
{
    *(dspp + 1) = (DSTKV)fputc((char)*(dspp + 1) , (FILE *)*dspp);
    dspp++;
}

static  void    f_wstr(void)
{
    *(dspp + 1) = fputs((char *)*dspp , (FILE *)*(dspp + 1));
    dspp++;
}

static  void    f_rchr(void)
{
    *dspp = fgetc((FILE *)*dspp);
}

static  void    f_rstr(void)
{
    if(fgets((char *)*(dspp + 2) , (int)*(dspp + 1) , (FILE *)*dspp) == NULL)
        *(dspp + 2) = 0;
    else
        *(dspp + 2) = 1;
    dspp += 2;
}

static  void    f_system(void)
{
    system((char *)*dspp++);
}

/* ----------------------------------------------------------------- */

jmp_buf     env;

static  void    f_longjump(void)
{
    longjmp(env , DIRECT_NORMALEND);
}

/* ----------------------------------------------------------------- */

static  void    f_usrwords(void);
static  void    f_syswords(void);
extern  void    f_defwords(void);

static  void    f_help(void)
{
    printf("*****  (userwords) *****¥n");
    f_usrwords();
    printf("***** (systemwords) *****¥n");
    f_syswords();
    printf("***** (definewords) *****¥n");
    f_defwords();
}

/* ----------------------------------------------------------------- */

static  void    RS_send_data(void)
{
    rs_send_data((unsigned char)*dspp++);
}

static  void    RS_receive_data(void)
{
    *(--dspp) = rs_receive_data();
}

static  void    RS_peek_data(void)
{
    *(--dspp) = rs_peek_data();
}

static  void    RS_send_buffer_space(void)
{
    *(--dspp) = rs_send_buffer_space();
}

static  void    RS_receive_count(void)
{
    *(--dspp) = rs_receive_count();
}

static  void    RS_send_count(void)
{
    *(--dspp) = rs_send_count();
```

パソコン通信用語集
ソフトウェアフロー制御　無手順通信でもっとも一般的なフロー制御。バッファがいっぱいになったときXON／OFFのコードを出力して知らせる。

```
}

static  void    RS_xcontrol_check(void)
{
    *(--dspp) = rs_xcontrol_check();
}

static  void    RS_dcd_check(void)
{
    *(--dspp) = rs_dcd_check();
}

static  void    RS_cts_check(void)
{
    *(--dspp) = rs_cts_check();
}

static  void    RS_dsr_check(void)
{
    *(--dspp) = rs_dsr_check();
}

static  void    RS_set_timer(void)
{
    rs_set_timer((int)*dspp++);
}

static  void    RS_chk_timer(void)
{
    *(--dspp) = rs_chk_timer();
}

/* --------------------------------------------------------------------- */

static  void    rs_set_baud_rate(void)
{
    rs_baud_rate = f_pop();
}

static  void    rs_set_data_length(void)
{
    rs_data_length = f_pop();
}

static  void    rs_set_parity_bit(void)
{
    rs_parity_bit = f_pop();
}

static  void    rs_set_stop_bit(void)
{
    rs_stop_bit = f_pop();
}

static  void    rs_set_xcontrol(void)
{
    rs_xcontrol = f_pop();
}

/* --------------------------------------------------------------------- */

struct  wtable  fname_table[] =
        {f_dup                  , "dup" ,
         f_truedup              , "?dup" ,
         f_drop                 , "drop" ,
         f_rdrop                , "rdrop" ,
         f_swap                 , "swap" ,
         f_rot                  , "rot" ,
         f_rrot                 , "rrot" ,
         f_over                 , "over" ,
         f_pick                 , "pick" ,
         f_cprint               , "cprint" ,
         f_sprint               , "sprint" ,
         f_bprint               , "bprint" ,
         f_wprint               , "wprint" ,
         f_nprint               , "nprint" ,
         f_qcinput              , "?cinput" ,
         f_cinput               , "cinput" ,
         f_sinput               , "sinput" ,
         f_slength              , "slen" ,
         f_exit                 , "exit" ,
         f_exitn                , "exitn" ,
         f_rsptop               , "i" ,
         f_rsp2nd               , "j" ,
         f_todsp                , "r>" ,
         f_torsp                , ">r" ,
         f_leave                , "leave" ,
         f_return               , "return" ,
         f_endword              , "]" ,
         f_longjump             , "$$longjump" ,
         f_add                  , "+" ,
         f_sub                  , "-" ,
         f_mul                  , "*" ,
         f_div                  , "/" ,
         f_and                  , "and" ,
         f_or                   , "or" ,
         f_xor                  , "xor" ,
         f_not                  , "not" ,
         f_cmp                  , "cmp" ,
         f_shl                  , "shl" ,
         f_shr                  , "shr" ,
         f_inc                  , "1+" ,
         f_dec                  , "1-" ,
         f_equ                  , "=" ,
         f_big                  , ">" ,
         f_sml                  , "<" ,
         f_load                 , "@" ,
         f_wload                , "w@" ,
         f_bload                , "b@" ,
         f_store                , "!" ,
         f_wstore               , "w!" ,
         f_bstore               , "b!" ,
         f_incstore             , "1+!" ,
         f_decstore             , "1-!" ,
         f_addstore             , "+!" ,
         f_substore             , "-!" ,
         f_scopy                , "s!" ,
         f_scat                 , "s+" ,
         f_scmp                 , "scomp" ,
         f_sequ                 , "s=" ,
         f_open                 , "fopen" ,
         f_close                , "fclose" ,
         f_wchr                 , "fcprint" ,
         f_wstr                 , "fsprint" ,
         f_rchr                 , "fcinput" ,
         f_rstr                 , "fsinput" ,
         f_system               , "system" ,
         f_syswords             , "systemwords" ,
         f_usrwords             , "userwords" ,
         f_defwords             , "definewords" ,
         f_help                 , "help" ,
/* ------------------------------------------------------------------------- */
         rs_initialize          , "rs_initialize" ,
         rs_finish              , "rs_finish" ,
         RS_send_data           , "rs_send_data" ,
         RS_receive_data        , "rs_receive_data" ,
         RS_peek_data           , "rs_peek_data" ,
         RS_send_buffer_space   , "rs_send_buffer_space" ,
         RS_receive_count       , "rs_receive_count" ,
         RS_send_count          , "rs_send_count" ,
         rs_xcontrol_on         , "rs_xcontrol_on" ,
         rs_xcontrol_off        , "rs_xcontrol_off" ,
         rs_rts_on              , "rs_rts_on" ,
         rs_rts_off             , "rs_rts_off" ,
         rs_dtr_on              , "rs_dtr_on" ,
         rs_dtr_off             , "rs_dtr_off" ,
         rs_break_on            , "rs_break_on" ,
         rs_break_off           , "rs_break_off" ,
         RS_xcontrol_check      , "rs_xcontrol_check" ,
         RS_dcd_check           , "rs_dcd_check" ,
         RS_cts_check           , "rs_cts_check" ,
         RS_dsr_check           , "rs_dsr_check" ,
         RS_set_timer           , "rs_set_timer" ,
         RS_chk_timer           , "rs_chk_timer" ,
/* ------------------------------------------------------------------------- */
         rs_set_baud_rate       , "rs_set_baud_rate" ,
         rs_set_data_length     , "rs_set_data_length" ,
         rs_set_parity_bit      , "rs_set_parity_bit" ,
         rs_set_stop_bit        , "rs_set_stop_bit" ,
         rs_set_xcontrol        , "rs_set_xcontrol" ,

/* ------------------------------------------------------------------------- */
         NULL        , "" };

/* =========================================================================
Cで記述された関数を検索する。見つかったら関数へのポインタを返す。見つから
ない場合はNULLを返す。これは一般ワード用。
========================================================================= */

void    (*exist_word(struct wtable *tptr , char *sptr))()
{
    for( ; tptr->wordptr != NULL ; tptr++)
    {
        if(strcmp(tptr->wordname , sptr) == 0)
            return  tptr->wordptr;
    }
    return  NULL;
}

void    (*exist_funcword(char *sptr))()
{
    return  exist_word(fname_table , sptr);
}

/* =========================================================================
データスタックダンプ。キー入力モード時のみ。
========================================================================= */

void    dump_dstack(void)
{
DSTKV   *dptr;

    printf("Stack dump : ");
    if(dspp > &dsp[DSP_MAX])
    {
        printf("(*** under flow ***)¥n");
        dspp = &dsp[DSP_MAX];
        return;
    }
    for(dptr = &dsp[DSP_MAX - 1] ; dptr >= dspp ; dptr--)
    {
        printf("%08lX ",*dptr);
    }
    puts("");
}

/* =========================================================================
Cで記述された関数をダンプする。
========================================================================= */

static  void    sysword_dump(struct wtable  *tptr)
{
    for( ; tptr->wordptr != NULL ; tptr++)
        printf("(%04X) %s¥n" , tptr->wordptr , tptr->wordname);
}

static  void    f_usrwords(void)
{
    sysword_dump(fname_table);
}

static  void    f_syswords(void)
{
extern  struct wtable cname_table[];
```


パソコン通信用語集
ダイヤルトーン　受話器をとったときに，使用可能を示す発信音のこと．モデムはこの発信音で回線がモデムに接続されているかを知ることができる．

```

    sysword_dump(cname_table);
}

/* ======================================================================
実行時インタプリタ
====================================================================== */

void    interp(struct pcode *enter)
{
struct  pcode *go;

    rs_baud_rate = 3;
    pcount.pptr = enter;
    for(;;)
    {
        if(pcount.pptr->ptype == OBJMODE_C)
        {
            ((pcount.pptr++)->pbody.fptr)();
        }
        else
        {
            go = pcount.pptr->pbody.pcptr;
            (--rspp)->stkptr = ++pcount.pptr;
            pcount.pptr = go;
        }
    }
}
```

リスト8 INTERP.H

```
void    f_push_string(void);
void    f_push_buffer(void);
void    f_push_wnumber(void);
void    f_push_lnumber(void);
void    f_doentry(void);
void    f_loopentry(void);
void    f_jmp(void);
void    f_truejmp(void);
void    f_falsejmp(void);
void    f_endword(void);
void    (*exist_word(struct wtable *tptr , char *sptr))();
void    (*exist_funcword(char *sptr))();
void    interp(struct pcode *);
```

リスト9 COMPILE.C

```
/* ======================================================================

    COMPILE.C          Copyright 1991 bY JUSTECH-LABO. (H.kanoshima)

====================================================================== */

#include    <stdlib.h>
#include    <stdio.h>
#include    <string.h>
#include    <jctype.h>
#include    <setjmp.h>

#include    "forthcom.h"
#include    "interp.h"
#include    "object.h"
#include    "sleap.h"

#define     DUMP    0

#define     KBYTE           * 1024
#define     OBJECT_SIZE     (10 KBYTE)

#define     LST_MAX         16      /* 分岐構文解析用スタックの要素数 */

#define     CHARSIZ         (sizeof(char))
#define     INTSIZ          (sizeof(int))

typedef     enum{LCODE_IF  , LCODE_DO , LCODE_BEGIN } LCODE;

/* ----------------------------------------------------------------------
シンボルテーブルはオブジェクトフィールドの最高位アドレスから低位アドレスに
向かって生成される。これはバッファを有効に使用するための処置である。シンボ
ルテーブルのポインタとオブジェクトフィールドのポインタの差が使用可能サイズ
である。シンボルのフォーマットは下記のとおり。なお、SYMflagはいまのところ使
用されていない。
---------------------------------------------------------------------- */

struct  symbol{
    struct  pcode *SYMbody;
    int     SYMflag;
    char    *SYMname;
    int     SYMnamelen;
};

/* ----------------------------------------------------------------------
オブジェクトフィールドを指すポインタと、オブジェクトフィールドのベースアド
レスを保持するポインタである。
---------------------------------------------------------------------- */

OBJ     objp , objbase , objpset;

/* ----------------------------------------------------------------------
シンボルテーブルを指すポインタと、シンボルテーブルのベースアドレスを保持す
るポインタである。
---------------------------------------------------------------------- */

struct  symbol  *symp , *symbase , *sympset;

/* ----------------------------------------------------------------------
分岐構文解析用のスタックとそのインデクスである。
---------------------------------------------------------------------- */

struct  {
    LCODE   LSTcode;
    OBJ     LSTbody;
}lstack[LST_MAX];

int     lst_idx = LST_MAX;

/* ======================================================================
バッファの初期化。オブジェクトフィールド（シンボルテーブルと共用）のメモリ
獲得と、リターン／データ／構文解析用の各スタックのインデクスを初期化してい
る。
====================================================================== */

BOOL    buffer_init(void)
{
    objp.cptr = objbase.cptr = malloc(OBJECT_SIZE);
    if(objp.cptr == NULL)   return  BFALSE;
    symp = symbase = (struct symbol *)(objbase.cptr + OBJECT_SIZE);

    return  BTRUE;
}

/* ======================================================================
エラー終了処理。
====================================================================== */

static  void    init_lstack(void)
{
    lst_idx = LST_MAX;
}

static  void    reset_symp_and_objp(void)
{
    objp = objpset;
    symp = sympset;
}

void    set_symp_and_objp(void)
{
    objpset = objp;
    sympset = symp;
}

extern      jmp_buf env;

void    exit_error(int error_code)
{
    if(is_directin() == BFALSE) exit(error_code);
    else
    {
        init_lstack();
        init_sourceline();
        reset_symp_and_objp();
        longjmp(env, 2);
    }
}

/* ======================================================================
コンパイル終了表示（キー入力時のみ）。
====================================================================== */

static  void    compile_mes(void)
{
    if(is_directin() == BFALSE)     return;
    printf("Compiled : %04X - %04X (%s)¥n", objpset, objp.cptr-1, symp->SYMname);
}

/* ======================================================================
分岐構文スタックへのアクセス関数。
```

パソコン通信用語集
端末速度固定　モデムポートの物理伝送速度に関係なくシリアルポートの物理伝送速度を一定にすること。この機能を使うことで端末側の負担を軽減することができる。

```
======================================================================= */

void    gen_func(void (*)());

static  void    push_lstack(LCODE lc , union pb *pbptr)
{
    if(lst_idx == 0)
    {
        print_source_and_line();
        printf("分岐構文スタックがいっぱいです¥n");
        exit_error(ERROR_OBJFULL);
    }
    lst_idx--;
    lstack[lst_idx].LSTcode = lc;
    lstack[lst_idx].LSTbody.bptr = pbptr;
}

static  void    check_lstack_empty(void)
{
    if(lst_idx == LST_MAX)
    {
        print_source_and_line();
        printf("分岐構文スタックが空です¥n");
        exit_error(ERROR_OBJFULL);
    }
}

static  void    check_lstack_notempty(void)
{
    if(lst_idx != LST_MAX)
    {
        print_source_and_line();
        printf("分岐構文がアンバランスです¥n");
        exit_error(ERROR_COMPILE);
    }
}

static  void    drop_lstack(void)
{
    check_lstack_empty();
    lst_idx++;
}

static  void    lstack_unbalance(void)
{
    print_source_and_line();
    printf("分岐構文がアンバランスです¥n");
    exit_error(ERROR_OBJFULL);
}

static  void    check_lcode(LCODE lc)
{
    check_lstack_empty();
    if(lstack[lst_idx].LSTcode != lc)   lstack_unbalance();
}

static  void    rewrite_jmpaddr(struct pcode *pbptr)
{
    check_lstack_empty();
    lstack[lst_idx].LSTbody.bptr->pcptr = pbptr;
}

static  void    rewrite_lstack(union pb *pbptr)
{
    check_lstack_empty();
    lstack[lst_idx].LSTbody.bptr = pbptr;
}

static  void    gen_loopback_addr(void)
{
    check_lstack_empty();
    objp.bptr->pcptr = lstack[lst_idx].LSTbody.pptr;
    objp.bptr++;
}

/* =====================================================================
ここはコンパイラが暗黙に使用する関数群である。コンパイル中に起動されてコン
パイルの一部を代行するもの（分岐構文のコンパイル）もある。これを追加定義し
た場合は必ずテーブルに登録しておくこと。
===================================================================== */

static void f_ifcomp(void (*fptr)())
{
    gen_func(fptr);
    push_lstack(LCODE_IF , objp.bptr);
    objp.bptr->pcptr = NULL;
    objp.bptr++;
}

static  void    f_if(void)
{
    f_ifcomp(f_falsejmp);
}

static  void    f_fif(void)
{
    f_ifcomp(f_truejmp);
}

static  void    f_else(void)
{
union   pb *oldpb;

    check_lcode(LCODE_IF);
    gen_func(f_jmp);
    objp.bptr->pcptr = NULL;
    oldpb = objp.bptr++;
    rewrite_jmpaddr(objp.pptr);
    rewrite_lstack(oldpb);
}

static  void    f_then(void)
{
    check_lcode(LCODE_IF);
    rewrite_jmpaddr(objp.pptr);
    drop_lstack();
}

static  void    f_do(void)
{
    gen_func(f_doentry);
    push_lstack(LCODE_DO , objp.bptr);
}

static  void    f_loop(void)
{
    check_lcode(LCODE_DO);
    gen_func(f_loopentry);
    gen_loopback_addr();
    drop_lstack();
}

static  void    f_begin(void)
{
    push_lstack(LCODE_BEGIN , objp.bptr);
}

static  void    f_until(void)
{
    check_lcode(LCODE_BEGIN);
    gen_func(f_truejmp);
    gen_loopback_addr();
    drop_lstack();
}

static  void    f_repeat(void)
{
    check_lcode(LCODE_BEGIN);
    gen_func(f_jmp);
    gen_loopback_addr();
    drop_lstack();
}

struct  wtable  cname_table[] =
        {f_if           , "if" ,
         f_fif          , "-if" ,
         f_else         , "else" ,
         f_then         , "then" ,
         f_do           , "do" ,
         f_loop         , "loop" ,
         f_begin        , "begin" ,
         f_until        , "until" ,
         f_repeat       , "repeat" ,
         NULL           , "" };

/* =====================================================================
ユーザ定義ワードを検索する。見つかったらそれへのポインタを返す。見つからな
い場合はNULLを返す。
===================================================================== */

static  struct  pcode   *exist_forthword(char *sptr)
{
struct  symbol  *symptr;
int     symlen;

    symlen = strlen(sptr);

    for(symptr = symp ; symptr < symbase ; symptr++)
    {
        if(symptr->SYMnamelen == symlen)
        {
            if(strncmp(symptr->SYMname , sptr , symlen) == 0)
                return  symptr->SYMbody;
        }
    }
    return  NULL;
}

/* =====================================================================
ユーザ定義ワードをダンプする。
===================================================================== */

void    f_defwords(void)
{
struct  symbol  *symptr;

    for(symptr = symp ; symptr < symbase ; symptr++)
        printf("(%04X) %s¥n",symptr->SYMbody , symptr->SYMname);
}

/* =====================================================================
Cで記述された関数を検索する。見つかったら関数へのポインタを返す。見つから
ない場合はNULLを返す。これはコンパイリングワード用。検索用関数の本体と一般
ワード用の検索関数は"INTERP.C"。
===================================================================== */

static  void    (*exist_compilingword(char *sptr))()
{
    return  exist_word(cname_table , sptr);
}

/* =====================================================================
オブジェクトフィールドの容量をチェックする。「これくらい空いているか」とい
う問い合わせである。空いていなければこの中で落ちる。
===================================================================== */

static  void    check_objectlimit(size_t size)
{
    if(((char *)symp - objp.cptr) <= size)
    {
        print_source_and_line();
        printf("オブジェクトバッファがいっぱいです¥n");
```


パソコン通信用語集
チェックサム（Check Sum）　誤り検出方法のひとつで，データを順にたしていった値

```
        exit_error(ERROR_OBJFULL);
    }
}

/* ===================================================================
オブジェクトフィールドにcharを1バイト生成する。
=================================================================== */

static  void    gen_char(unsigned char c)
{
    check_objectlimit(sizeof(char));
    *objp.cptr++ = c;
}

/* ===================================================================
オブジェクトフィールドにint またはlongを生成する。
=================================================================== */

static  void    gen_int(unsigned int i)
{
    check_objectlimit(sizeof(int));
    *objp.iptr++ = i;
}

static  void    gen_long(long l)
{
    check_objectlimit(sizeof(int));
    *objp.lptr++ = l;
}

/* ===================================================================
オブジェクトフィールドに関数アドレスのpcode を生成する。
=================================================================== */

static  void    gen_func(void (*func)())
{
    check_objectlimit(sizeof(struct pcode));
    objp.pptr->ptype = 'C';
    objp.pptr->pbody.fptr = func;
    objp.pptr++;
}

/* ===================================================================
オブジェクトフィールドにユーザ定義ワードのpcode を生成する。
=================================================================== */

static  void    gen_word(struct pcode *word)
{
    check_objectlimit(sizeof(struct pcode));
    objp.pptr->ptype = 'W';
    objp.pptr->pbody.pcptr = word;
    objp.pptr++;
}

/* ===================================================================
オブジェクトフィールドに文字列を生成する。ここで生成されるのは文字列の本体
のみである。
=================================================================== */

static  void    gen_string(char *sptr)
{
int     slen;

    slen = strlen(sptr) + 1;
    check_objectlimit(slen);
    strcpy(objp.cptr , sptr);
    objp.cptr += slen;
}

/* ===================================================================
オブジェクトフィールドに指定容量のバッファを生成する。int 値のバッファサイ
ズとバッファ本体のスペースが生成され、バッファ内部は0でクリアされる。
=================================================================== */

static  void    gen_buffer(int bufsz)
{
    gen_int(bufsz + sizeof(int));
    check_objectlimit(bufsz);
    while(bufsz--)  *objp.cptr++ = 0;
}

/* ===================================================================
オブジェクトフィールドに文字列をスタックに積むオブジェクトを生成する。専用
の関数と文字列のサイズ（末尾のヌルも含めたバイト数）、および文字列の本体が
生成される。
=================================================================== */

static  void    compile_string(char *sptr)
{
    gen_func(f_push_string);
    gen_int(strlen(sptr)+sizeof(int)+1);
    gen_string(sptr);
}

/* ===================================================================
オブジェクトフィールドに数値をスタックに積むオブジェクトを生成する。専用の
関数とint 値の数値が生成される。
=================================================================== */

static  void    compile_number(DSTKV value)
{
    if((value >= 0) && (value <= 0xffff))
    {
        gen_func(f_push_wnumber);
        gen_int((int)value);
    }
    else
    {
        gen_func(f_push_lnumber);
        gen_long(value);
    }
}

/* ===================================================================
二重定義エラーを表示して落ちる。
=================================================================== */

static  void    double_def_error(char *sptr)
{
    print_source_and_line();
    printf("'%s' は二重定義です¥n", sptr);
    exit_error(ERROR_DOUBLEDEF);
}

/* ===================================================================
コンパイルエラーを表示して落ちる。
=================================================================== */

static  void    undefined_error(char *sptr)
{
    print_source_and_line();
    printf("'%s' は未定義のワードです¥n", sptr);
    exit_error(ERROR_DEFINE);
}

/* ===================================================================
シンボルテーブルを作成する。二重定義は内部で落ちる。
=================================================================== */

static  void    gen_symbol(char *sptr)
{
    if(exist_forthword(sptr) != NULL)       double_def_error(sptr);
    if(exist_compilingword(sptr) != NULL)   double_def_error(sptr);
    if(exist_funcword(sptr) != NULL)        double_def_error(sptr);

    symp--;
    symp->SYMflag = 0;
    symp->SYMname = objp.cptr;
    gen_string(sptr);
    symp->SYMnamelen = strlen(sptr);
    symp->SYMbody = objp.pptr;
}

/* ===================================================================
1単語をコンパイルする。すでにシンボルは生成されている。']' をコンパイルし
たときのみBFALSEを、それ以外の場合はBTRUEを返す。
=================================================================== */

static  long    nconvert(char *sptr)
{
char    *conv;

        if(*sptr == '$')    return  strtol(sptr+1 , &conv , 16);
        else                return  strtol(sptr   , &conv , 10);
}

static  BOOL    compile_1word(IDCODE id , char *sptr)
{
struct  pcode   *eobj;
void    (*fobj)();

    if(id == ID_ENDLABEL)
    {
        gen_func(f_endword);
        return  BFALSE;
    }
    if(id == ID_NUMBER)
    {
        compile_number(nconvert(sptr));
        return  BTRUE;
    }
    else if(id == ID_CHAR)
    {
    int     value = 0;

        value = *sptr++;
        if(*sptr != '¥0')
        value += *sptr << 8;
        compile_number(value);
        return  BTRUE;
    }
    else if(id == ID_STRING)
    {
        compile_string(sptr);
        return  BTRUE;
    }
    else if(id == ID_WORD)
    {
        if((eobj = exist_forthword(sptr)) != NULL)
        {
            gen_word(eobj);
            return  BTRUE;
        }
        else if((fobj = exist_funcword(sptr)) != NULL)
        {
            gen_func(fobj);
            return  BTRUE;
        }
        else if((fobj = exist_compilingword(sptr)) != NULL)
        {
            (*fobj)();
            return  BTRUE;
        }
        else
            undefined_error(sptr);
    }
    else
        undefined_error(sptr);
    return  BFALSE;
}

/* ===================================================================
```

パソコン通信用語集
データ長　1バイトの文字を表現するには8ビットのデータが必要となる。しかし，アルファベットだけなら7FHまでの7ビットで表現できる。

```
ユーザ定義単語のコンパイル。定義終了までこの中でループする。
=================================================================== */

void    definition_word(void)
{
char    wordbody[WORDLINE_MAX];
IDCODE  id;

    id = get_word(wordbody);
    if(id != ID_WORD)
    {
        print_source_and_line();
        printf("指定できないワードを定義指定しました - '%s'Yn",wordbody);
        exit_error(ERROR_DEFINE);
    }
    else
    {
        gen_symbol(wordbody);
        do
        {
            id = get_word(wordbody);
        }while(compile_1word(id , wordbody) != BFALSE);
        check_lstack_notempty();
    }
    compile_mes();
}

/* ===================================================================
変数の定義。
=================================================================== */

void    definition_var(void)
{
char    wordbody[WORDLINE_MAX];
IDCODE  id;

    id = get_word(wordbody);
    if(id != ID_WORD)
    {
        print_source_and_line();
        printf("指定できないワードを変数指定しました - '%s'Yn",wordbody);
        exit_error(ERROR_DEFINE);
    }
    else
    {
        gen_symbol(wordbody);
        gen_func(f_push_buffer);
        gen_buffer(sizeof(DSTKV));
        gen_func(f_endword);
    }
    compile_mes();
}

/* ===================================================================
バッファの定義。
=================================================================== */

void    definition_buffer(void)
{
char    wordbody[WORDLINE_MAX];
IDCODE  id;
int     bufsz;

    id = get_word(wordbody);
    if(id != ID_WORD)
    {
        print_source_and_line();
        printf("指定できないワードをバッファ指定しました - '%s'Yn",wordbody);
        exit_error(ERROR_DEFINE);
    }
    else
    {
        gen_symbol(wordbody);
    }
    id = get_word(wordbody);
    if(id != ID_NUMBER)
    {
        print_source_and_line();
        printf("サイズに数値以外のものを指定しました - '%s'Yn",wordbody);
        exit_error(ERROR_DEFINE);
    }
    else
    {
        gen_func(f_push_buffer);
        bufsz = nconvert(wordbody);
        gen_buffer(bufsz);
        gen_func(f_endword);
    }
    compile_mes();
}

/* ===================================================================
ワード実行指定。
=================================================================== */

void    execute_word(void)
{
static  char    wordbody[WORDLINE_MAX];
IDCODE  id;
struct  pcode *enter;

    id = get_word(wordbody);
    if(exist_forthword(wordbody) == NULL)
    {
        print_source_and_line();
        printf("実行できないワードを実行指定しました - '%s'Yn",wordbody);
        exit_error(ERROR_COMPILE);
    }
    else
    {
        enter = objp.pptr;
        compile_1word(id , wordbody);
        compile_1word(ID_WORD , "exitn");
        interp(enter);
    }
}

/* ===================================================================
即実行モード。即実行は']'で終端する。
=================================================================== */

void    direct_execution(IDCODE id , char *wptr)
{
char    wordbody[WORDLINE_MAX];
struct  pcode *enter;

    enter = objp.pptr;
    if(compile_1word(id , wptr) == BFALSE)  return;
    cat_sourceline(ENDWORD);
    for(;;)
    {
        id = get_word(wordbody);
        if(id == ID_ENDLABEL)   break;
        compile_1word(id , wordbody);
    }
    check_lstack_notempty();
    compile_1word(ID_WORD , "$$longjump");
    objp.pptr = enter;
    interp(enter);
}

#if     DUMP
/* ===================================================================
コンパイル後のレポート表示（デバッグ用）。オブジェクトバッファとシンボルバッ
ファをダンプする。
=================================================================== */

void    dump(char *start , char *end)
{
int     i = 0;
char    ascbuf[32] , *ascbufp , *s , *e;

    if(start >= end)    return;
    printf("Yn[dump %04X - %04X]Yn",start , end);
    printf("Addr   +0 +1 +2 +3 +4 +5 +6 +7 - +8 +9 +A +B +C +D +E +F ");
    printf("/ 0123456789ABCDEFYn");
    printf("--------------------------------------------------------");
    printf("-----------------Yn");
    s = (char *)((int)start & 0xfff0);
    e = (char *)(((int)end & 0xfff0)+15);
    while(s <= e)
    {
        printf("%04X - ",s);
        ascbufp = ascbuf;
        for(i = 0 ; i < 16 ; i++)
        {
            if((s < start) || (s > end))
            {
                printf("   ");
                *ascbufp++ = ' ';
            }
            else
            {
                printf("%02X ",*s);
                if(*s >= ' ')   *ascbufp++ = *s;
                else            *ascbufp++ = '.';
            }
            if(i == 7)  printf("- ");
            s++;
        }
        *ascbufp = 'Y0';
        printf("/ %s Yn",ascbuf);
    }
}

void    report(void)
{
    printf("OBJ = %04X/%04X(%04d Bytes) : SYM = %04X/%04X(%04d Bytes)Yn",
            objp,objbase,objp.cptr-objbase.cptr,
            symp,symbase,sizeof(struct symbol)*(symbase-symp));
    dump(objbase.cptr , objp.cptr-1);
    dump((char *)symp , (char *)symbase-1);
}

#else

void    report(void)
{
}

#endif
```


パソコン通信用語集
デコード　エンコードに対して，テキストファイルに変換されたバイナリファイルを元のファイルに戻すこと。

区別をつけるために「.S」という拡張子をつけておくことをお勧めする．このファイルをSLEAPでコンパイルするには2通りの方法がある．

図8の例は，SLEAPを単純起動してから単語「compile」を使ってコンパイルする方法である．コンパイルした単語が次々と表示されていく．

SLEAPを起動するときにオプションとしてファイル名を指定すると，起動後にコンパイルをはじめる（図9）．この場合には，コンパイルした単語名は表示しない仕様にしてある．コンパイルさせたいファイルは空白で区切って複数指定することができる．

コンパイルとリンク

さて，今度はSLEAP自身のコンパイルとリンクの話である．SLEAPは3本のCソースからなる．それらをコンパイルして，2章の通信ライブラリとリンクする．

筆者が使用したコンパイラは，ボーランド社のBorland C++ Version 2.00である．リスト13のようなMAKEファイルを作って，コンパイルすることにする．

コンピュータ同士を繋いでみよう

通信言語コンパイラSLEAPのご紹介が終ったところで，いよいよ実践

LIST

リスト10 COMPILE.H

```
BOOL    buffer_init(void);
void    set_symp_and_objp(void);
void    exit_error(int);
void    definition_word(void);
void    definition_var(void);
void    definition_buffer(void);
void    execute_word(void);
void    direct_execution(IDCODE , char *);
void    report(void);
void    dump_dstack(void);
```

リスト11 FORTHCOM.H

```
#define     OP_PRIM         0
#define     OP_WORD         1
#define     OP_ENDW         2
#define     OP_PRIM_ENDW    (OP_PROM + OP_ENDW)
#define     OP_WORD_ENDW    (OP_WORD + OP_ENDW)

#define     ERROR_NOFILE        1
#define     ERROR_OBJFULL       2
#define     ERROR_DOUBLEDEF     3
#define     ERROR_DEFINE        4
#define     ERROR_COMPILE       5
#define     ERROR_SOURCEMAX     6
#define     DIRECT_NORMALEND    7

#define     SRCLINE_MAX     1024
#define     WORDLINE_MAX    128

#define     ENDWORD         "]"

typedef     enum    {BFALSE,BTRUE}  BOOL;

typedef     enum {  ID_LABEL , ID_VAR    , ID_BUFFER , ID_END ,
                    ID_WORD  , ID_NUMBER , ID_STRING , ID_CHAR ,
                    ID_COMPILE , ID_ENDLABEL    }   IDCODE;
```

リスト12 OBJECT.H

```
#define     OBJMODE_C       'C'     /* Cで記述されたオブジェクト */
#define     OBJMODE_W       'W'     /* ＦＯＲＴＨ記述のオブジェクト */

/* ---------------------------------------------------------------------
"pcode"は本コンパイラの基本オブジェクトフォーマットである。その構成は下記の
とおり、char型の識別コード（"OBJMODE_C" / "OBJMODE_W"）と、Cで記述された関
数へのポインタもしくは別のpcodeへのポインタである。
--------------------------------------------------------------------- */

union   pb{
        struct  pcode *pcptr;
        void    (*fptr)();
};

struct  pcode{
    char    ptype;
    union   pb  pbody;
};

/* ---------------------------------------------------------------------
オブジェクトフィールドには上記のpcodeのほかに、文字列や数値などのさまざまな
データが混在しているので、それらのいずれかを指すポインタを定義するために下記
の共用体を定義する。
--------------------------------------------------------------------- */

typedef union   obj{
    char            *cptr;
    unsigned int    *iptr;
    long            *lptr;
    struct  pcode   *pptr;
    union   pb      *bptr;
} OBJ;

/* ---------------------------------------------------------------------
リターン／データ両スタックで管理されるデータのタイプを定義している。
--------------------------------------------------------------------- */

typedef     long    DSTKV;

typedef union   stkv{
        DSTKV   stknum;
        struct  pcode   *stkptr;
}   RSTKV;

/* ---------------------------------------------------------------------
Cで記述された関数テーブルのフォーマット。
--------------------------------------------------------------------- */

struct  wtable{
    void    (*wordptr)();
    char    *wordname;
};
```

編の本番となる．まずは，２台のコンピュータ同士をつないでのターミナルプログラムを作ってみよう．幸いにして98を２台お持ちの読者は実際に試していただきたい．

いままでに紹介した機能（通信ライブラリとSLEAP本体）で，ターミナルプログラムを書くことはもちろん可能であるが，このあとでネットワークへの接続プログラムなども作る関係上，ここで通信に必要なおおかたの単語を仕上げてしまうことにする．SLEAP

COLUMN

フロー制御
「待ってて」宣言をするための約束ごと

誰かと電話をしていて，相手が約束の場所や電話番号を話しだしたとき，ふと見ると手元にメモ用紙がない．だまって取りにいってしまうと相手はそんな事情はご存じないので話し続けてしまう．こんなとき，普通は「あ，ちょっと待っててね．メモ用紙取ってくるから」などといって相手に待ってもらうことだろう．

パソコンもRS-232Cからデータを受け取っていて，なにかの事情でデータを取ることができなくなり（メモ用紙を取りに行くことはあまりないようである．ディスクアクセスなどほかの処理で忙しい場合や単純に処理が遅くて受信が追いつかなくなるなどの理由によるものだろう），「ちょっと待っててね」が必要になることがある．RS-232Cによる通信では口でいうことはできないので，「待ってて」宣言をするための約束ごとをあらかじめ取り決めておかなければならない．この約束ごとをフロー制御と呼んでいる．

代表的なフロー制御に「Xコントロール」と「RS/CS制御」がある．ここでは，この２つのおおまかな原理を紹介しよう．

■Xコントロール

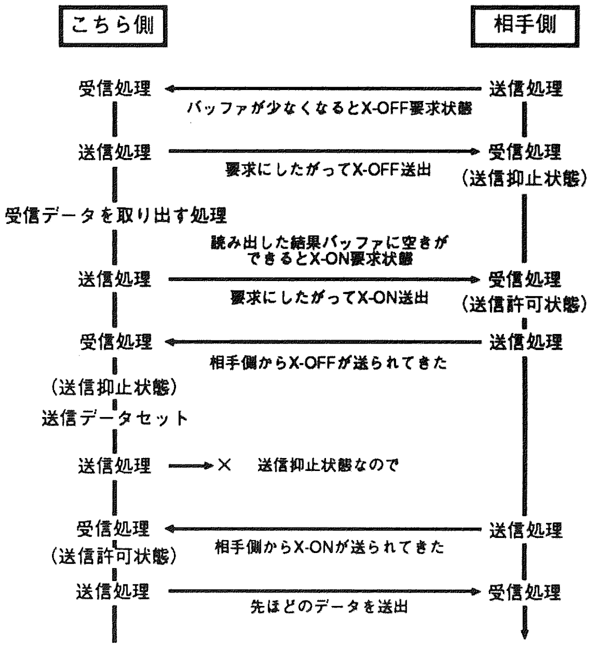
TYPEコマンドでテキストを画面表示して見ているとき，[CTRL]+[S]というキー操作で表示を一時停止にしたり，[CTRL]+[Q]で再開したりできることはご存じであろう．X制御はまさにそれの通信版である．相手側にデータを送るのを一時停止してもらいたいときに[CTRL]+[S]というデータを送出し（これをX-OFFと呼ぶ），再開してもらいたいときには[CTRL]+[Q]を送るのである（こちらがX-ONである）．逆の立場で考えてみれば，[CTRL]+[S]が送られてきたら，次に[CTRL]+[Q]がくるまで送信を保留する．

かんじんなのは，双方がこのルールをきちんと守ることである．通信は相手あってのことであるから，こちら側でいくら一生懸命「待っててね」といっても相手がそれを無視したり，最初からその意味をよくわかっていないとXコントロールは崩壊するのである．

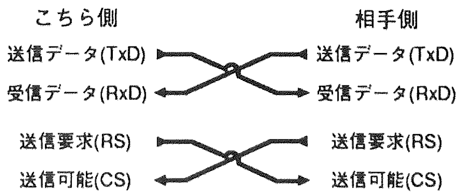
では，Xコントロールの流れをプログラムを書く立場からもう一度整理してみることにしよう．

●こちら側の受信バッファがあふれそうになった場合

・受信バッファの残量が設定値より少なくなった場合，X-OFF（[CTRL]+[S]＝13h）を相手側に送出する処理を行う．



図A Xコントロールつきの送受手順



図B

パソコン通信用語集
トランスペアレント（透過性）　すべてのコードをデータとして伝送できること．X-MODEMやY-MODEMはトランスペアレントなプロトコルといえる．

は，複数のマクロプログラムを次々と積み上げていくことができるので，ここで作ったものをそのまま応用してネットワークへのアクセスプログラムとするわけだ．

FORTHだ，スタックだ，基本単語だと少々地味な話題が続いてきたが，もう少しの辛抱である．マクロプログラムは次第に積み重なっていくと，非常に強力な通信用単語になっていく．そうなるとあとは楽なものなのだから．

・受信バッファからデータを取り出して，残量が設定値より多くなった時点でX-ON（[CTRL]＋[Q]＝11h）を送出する．

●相手側の受信バッファがあふれそうになった場合

・相手側からX-OFFが送出されてきたら，こちら側からの送信を停止する．

・相手側からX-ONが送出されてきたら，こちら側からの送信を再開する．

Xコントロールは，送受するデータ自体が制御のためのキーワードとなっているので，双方の呼吸が合えば特別なハードウェアが必要ない．ただし当然のことながら制御のために使用するコード（Xコントロールの場合は11hと13h）はデータとして使用することはできない．このため，Xコントロールをする場合はデータとして[CTRL]＋[S]や[CTRL]＋[Q]が入っていないもの，つまり現実にはテキストデータの送受にしか使えないことになる．

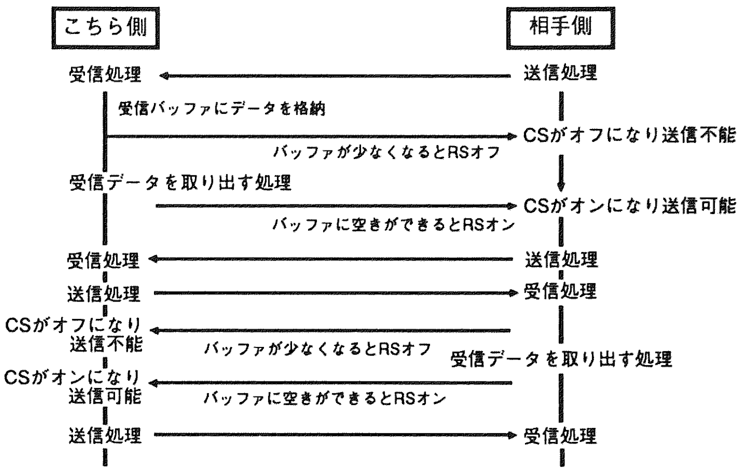
■RS/CS制御

RS-232Cのコネクタを見ると穴が25個あって，ケーブルを繋ぐとそれぞれの穴にピンが入って接続される．この25本のピンがおのおの規定された信号となる．詳細は後節に譲るとして，ここではその中の4つの信号について図Bのような結線をした場合を考えてみる．

お互いの送信データが受信データに，受信データが送信データになるのは当然として，問題は「送信要求」と「送信可能」である．この信号は，こちら側が送信を要求している（つまりこちら側が受信可能な状態である）ことを相手側に伝達するための信号である．

図Bを見ると，こちら側のRS端子が相手側のCS端子につながっている．こちら側でRSの信号を変化させることによって相手側のCS端子が変化することになる．RS/CS制御は，お互いの端子がこういう結線になっていることを前提として行われる．受信が可能な状態であるかぎり，RS端子をオンにしておき（相手側から見ると，送信可能がオンになっている状態），受信ができない状況になったときにRSをオフにする．このことによって相手側では送信可能がオフになり，送信が止まる．

では，RS/CS制御の流れをプログラムを書く立場からもう一度整理してみることにしよう．



図C
RS/CS制御
つきの送受信手順

■RS/CS制御の流れ

●こちら側の受信バッファがあふれそうになった場合

・受信バッファの残量が設定値より少なくなった場合，RSをオフにする処理を行う．

・RSをオフにすると，相手側のCSがオフになり送信ができなくなるので，結果的にデータが送られてこなくなる．

・受信バッファからデータを取り出して，残量が設定値より多くなった時点でRSをオンにする．相手側でCSがオンになるので送信が再開され，データが送られてくるようになる．

●相手側の受信バッファがあふれそうになった場合

・相手側でRSをオフにした場合は，こちら側のCSがオフになり送信ができなくなる．この結果送信割り込みが発生しなくなる．こちらの場合は，送信がハード的に止まるだけなので，ソフトウェアでは関知する必要がない．

RS/CS制御の処理は以上である．意外に簡単なものだと思われるだろう．専用の信号線を用いて制御を行うため，送受するデータにも制限はない．ただ，RS/CS制御は物理的な信号線を使用するので，この信号線がハード的に使用可能な状態になっていなければ使えない．とくに，ケーブルの結線が落とし穴になることもあるので注意が必要である．

パソコン通信用語集

ハードウェアフロー制御　バッファ機能のあるモデムで，端末とモデムを結ぶRS-232CインターフェースのRS（RTC）／CS（CTS）信号を使って行うフロー制御．

通信用マクロプログラム「COMLIB.S」

SLEAPのもつ基本単語の中には，通信ライブラリを呼び出す単語が含まれている．これを使って通信ソフトを組むことも可能ではあるが，単機能な基本単語をそのつど組み合わせていくのではたいへんである．そこで，通信

リスト 13 MAKEFILE

```
LINK_TARGET =    sleap.obj compile.obj interp.obj rs232LIB.obj

sleap.exe : $(LINK_TARGET)
        tlink c0s $(LINK_TARGET), sleap, sleap, cs

sleap.obj      : forthcom.h compile.h sleap.c
        bcc -c  sleap.c

compile.obj    : forthcom.h interp.h object.h sleap.h compile.c
        bcc -c  compile.c

interp.obj     : forthcom.h sleap.h object.h interp.c rs232lib.h
        bcc -c  interp.c

rs232LIB.obj   : rs232LIB.asm
        tasm /ML rs232LIB;
```

リスト 14 COMLIB.S

```
; ============================================================
; COMLIB.S
;
; Communication libraly for SLEAP system
;
; Create : 1991.10.12
;
;
; Copyright 1991 By Justech Labo. (H.kanoshima)
;
; ============================================================

;====== 初期化用パラメータ設定値 =====================

;====== ボーレート定数 =====================
[ B75                    0 ]
[ B150                   1 ]
[ B300                   2 ]
[ B600                   3 ]
[ B1200                  4 ]
[ B2400                  5 ]
[ B4800                  6 ]
[ B9600                  7 ]
[ B19200                 8 ]

; ====== キャラクタ長定数 =================
[ L7                     2 ]
[ L8                     3 ]

; ====== パリティビット定数 ===============
[ PNONE                  0 ]
[ PODD                   1 ]
[ PEVEN                  3 ]

; ====== ストップビット長定数 =============
[ S1                     1 ]
[ S15                    2 ]
[ S2                     3 ]

; ====== X-ON/OFF定数 =====================
[ XCON                   1 ]
[ XCOPF                  0 ]

[ set_default_comspec
        B2400   rs_set_baud_rate
        L8      rs_set_data_length
        PNONE   rs_set_parity_bit
        S1      rs_set_stop_bit
        XCON    rs_set_xcontrol ]

[ rs_send_cr    $0d rs_send_data ]
[ rs_send_lf    $0a rs_send_data ]

[ rs_send_crlf
        rs_send_cr
        rs_send_lf ]

[ rs_send_string        ; string -> (none)
        dup slen
        do
                dup b@ rs_send_data
                1+
        loop
        drop
        begin
                rs_send_count 0 =
                if      return
                then
        repeat ]

[ rs_send_string_cr     ; string -> (none)
        rs_send_string
        rs_send_cr ]

[ ?rsinput              ; (none) -> count
        rs_receive_count ]

[ rs_receive_data_wait  ; (none) -> char
        begin
                ?rsinput
                if      rs_receive_data return
                then
        repeat ]

[ cr_to_crlf    ; (none) -> (none)
        $0d = if $0a cprint then ]

var     rts_ff

[ rts_toggle    ; (none) -> (none)
        rts_ff @
        if      rs_rts_on 0 "rts_on¥n" sprint
        else    rs_rts_off 1 "rts_off¥n" sprint
        then
        rts_ff ! ]

var     xcont_ff

[ xcont_toggle  ; (none) -> (none)
        xcont_ff @
        if      $11 rs_send_data 0 "CTRL-Q¥n" sprint
        else    $13 rs_send_data 1 "CTRL-S¥n" sprint
        then
        xcont_ff ! ]

[ CTRL-R        $12 ]
[ CTRL-X        $18 ]
[ CTRL-Z        $1A ]

[ term
        set_default_comspec
        rs_initialize
        begin
                ?cinput
                        if      cinput dup CTRL-Z =
                                if      drop rs_finish exitn
                                else    dup CTRL-R =
                                        if      drop rts_toggle
                                        else    dup CTRL-X =
                                                if      drop xcont_toggle
                                                else    dup rs_send_data
                                                        dup cprint
                                                        cr_to_crlf
                                                then
                                        then
                                then
                        then
                ?rsinput
                        if      rs_receive_data dup cprint
                                cr_to_crlf
                        then
        repeat ]

"Compiled -comlib.s-¥n" sprint
```


パソコン通信用語集
バイナリファイル　00HからIFHの中にはコントロールコードと呼ばれる制御用文字コードが含まれる．バイナリファイルはこれらを含むので直接送ることはできない．

リスト 15 STDLB.S

```
; ============================================================
; STDLIB.S
;
; Standard libraly for SLEAP system
;
; Create : 1991.10.12
;
;
; Copyright 1991 By Justech Labo. (H.kanoshima)
;
; ============================================================

[ crlf  $0d cprint $0a cprint ]

[ npr nprint crlf ]

[ wpr wprint crlf ]

[ @npr  @ npr ]

[ 2drop drop drop ]

[ 3drop 2drop drop ]

[ 4drop 2drop 2drop ]

"Compiled -stdlib.s-¥n" sprint
```

表2 COMLIB.S に含まれる単語

set_default_comspec	入力：なし 出力：なし	通信条件をこの単語のもつデフォルトの設定とする．この単語は，通信ライブラリのもっている初期化パラメータエリアに値を設定するだけであって，実際のハード的な初期化はしない．設定する条件は次のとおりである．ほとんどのネットワークは，この設定でそのままアクセスすることができるが，デフォルトの設定を変えたい場合はこの単語を変更することになる． ボーレート　：2400 キャラクタ長　：8ビット パリティ種別　：なし ストップビット長：1ビット Xコントロール　：あり
rs_send_crlf	入力：なし 出力：なし	RS-232C に復帰コード（CR＝0x0D）と改行コード（LF＝0x0A）を送出する．
rs_send_string	入力：文字列をスタックに積む 出力：なし	スタックに積んだ文字列を送出する．文字列を通信ライブラリに引き渡した後，実際にすべてのデータが RS-232C から送出されるのを監視してから処理を終了する．
rs_send_string_cr	入力：文字列をスタックに積む 出力：なし	上記の文字列送出を行ったあとで，改行コードを送出する．
?rsinput	入力：なし 出力：受信済みデータバイト数がスタックに積まれる	受信済みが何バイトあるかをスタックに返す．受信データがない場合には当然0が返る．
rs_receive_data_wait	入力：なし 出力：受信データがスタックに積まれる	受信データをスタックに返す．実行時に受信データがなければ，受信があるまでこの単語の中で待ちつづける．
cr_to_crlf	入力：1文字をスタックに積む 出力：なし	スタックに積まれた文字が CR であれば，LF を表示する．ネットワークからの受信データで，行の終りに CR のみが付けられていることがあるので，そのまま表示すると改行しない．そういう場合にこの単語を使って受信側で改行処理を補う．
rts_toggle	入力：なし 出力：なし	この単語は，呼ばれるごとに RTS の ON と OFF を交互に設定する．設定したあとの状態は「rts_on」「rts_off」というように画面表示する．
xcont_toggle	入力：なし 出力：なし	この単語は，呼ばれるごとにXコントロールのための X-ON と X-OFF を交互に設定する．Xコントロールを使用するためには，初期化のときに「Xコントロールあり」にしなければならないことはもちろんである．設定した状態は「CTRL-Q」「CTRL-S」というように画面表示する． ここまでが，COMLIB.S の通信用単語である．これらを組み合わせて，通信ソフトを作る実例として，マクロプログラムの最後に「term」という単語を定義する．「term」は，必要最低限の通信機能をもった文字どおりのターミナルプログラムである．
term	入力：なし 出力：なし	「set_default_comspec」と「rs_initialize」を実行して，RS-232C の初期化を行い，受信があればそのデータを画面表示し，キー入力があればその文字を送信する．これの繰り返しをしているだけである．ただし以下のキー操作は特別に処理をする． CTRL＋X：操作するごとに X-ON と X-OFF を交互に送出する CTRL＋R：操作するごとに RTS-ON と RTS-OFF を交互に設定する CTRL＋Z：ターミナルの実行を終了する

パソコン通信用語集
バッファ　エラー訂正やデータ圧縮機能をもつモデムはシリアルポートとモデムポートの伝送速度が等しくない，この速度差を吸収するのがモデムバッファの役目である．

ソフトを作る上で汎用的に使用することのできるいくつかの処理単語をあらかじめ用意しておく．

たとえば，基本単語の1文字送信を使っての「1行送信」や「受信があるまで待って，データを取り込む」といった処理である．通信条件を変更することが少ないなら，その固定された通信条件で初期化をしてしまうという単語も便利だろう．このような単語集を「COMLIB.S」という名前で作っておく．この単語集に含まれる単語は表2のとおりである．

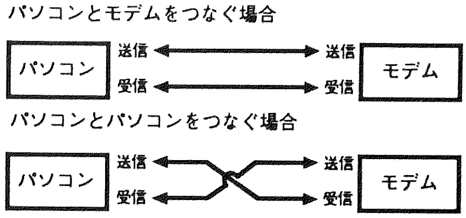


図10

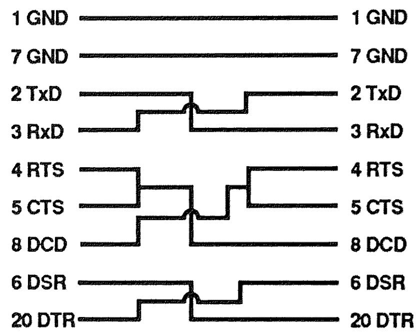


図11

結線! RSケーブルの陣

ターミナルプログラムが完成した．さて試験ということになったが，ここでなくてはならないものはRS-232Cのケーブルである．モデムでの試験であれば，モデム付属のケーブルがあるだろうから話は簡単だが，パソコン同士をつないで試験をしようと思うとケーブルをどこからか仕入れてこなければならない．意外なことにモデム付属のケーブルは，パソコン同士の接続には使うことができないのである．簡単にいってしまうと，図10のような接続ケーブルが必要なのである．

ようするに，パソコンとモデムの場合には，「送信と送信」，「受信と受信」がまっすぐに接続され(これをストレートケーブルと呼ぶ)，パソコンとパソコンをつなぐ場合には，こちら側の送信とあちら側の受信，こちら側の受信とあちら側の送信が互いに交差する形で接続されるのである(これをクロスケーブルと呼ぶ)．

モデムに付属してきたケーブルでは，パソコン同士の接続試験ができないことがわかった．それでは，そのクロスケーブルとかいうのを買ってこようと思ってパソコンのカタログをみるとこれが驚くほど高い．首都圏の方であれば，安いケーブルを売っている店をご存じかもしれないが，筆者の生活している地方都市ではそういう店もあまりない．かといってプログラムを書いたがテストはしていないではお天道様と

```
***** SLEAP ver 1.0 Copyright Justech labo. *****

Compiled -stdlib.s-
Compiled -comlib.s-
>> term                                ←単語「term」を実行する
test test test
^Z                                     ←終了（実際には画面表示されない）

A>
```

図12

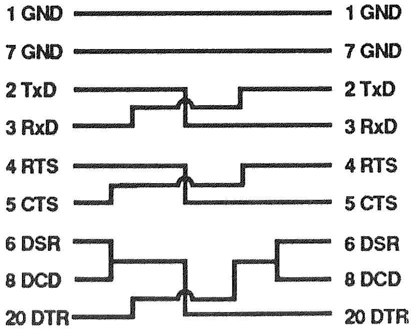


図13

パソコン通信用語集
パリティチェック　データの最後にパリティビットと呼ばれる誤り検出用のビットを用意し，送られたデータのビットの和が偶数か奇数かでチェックする．

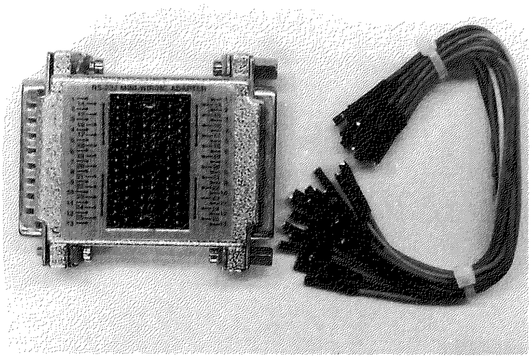


写真1
RS-232Cミニ ワイヤリングアダプタ

編集部が許してくれないだろうから，ここはケーブルの自作ということにあいなった．

RS-232Cのコネクタ（オス）を2個とケーブルを買ってくる．結線図を見ると，ケーブルは25芯ある必要はないので，10芯のものを2メートルほど買ってきた．結線図を見ながらピンを1つずつハンダ付けしていく．やること自体は単純なものなのだが，ピンが細かいのでなかなかたいへんである．慣れないハンダゴテをふり回し，腕に数か所のやけど跡を残し，どうにか完成した．

結線は，参考書を見て図11のように行った．

さっそく2台のパソコンをできたてのケーブルでつなぎ，今回のターミナルソフトを両方から起動してみる．起動方法は以下のとおりである．

```
A>SLEAP STDLIB.S COMLIB.S
```

SLEAPが起動すると図12のような画面になる．

1台のマシンのキーボードから文字をぽつぽつと入力していくと，そのとおりにもう1台のマシンの画面に表示が出てくるではないか．あたりまえのようだが，これがやけどの痛みも忘れるくらい感動してしまうものなのだ．FEPを起動すれば，漢字さえも送受できる．これで立派な(？)通信ソフトの誕生である．

まぁ，いい年をした男がパソコンの前でニヤニヤしながらキーボードを叩いては隣のパソコンのディスプレイを上体を倒して覗きこんでいるという図はあまり人に見られないほうがいいので，今度は別の機能をためしてみることにしよう．

通信ライブラリにあるフロー制御が実際にどう動くのかを実験する．termにはその実験用の機能を内蔵しているので，これを使うことにする．

【Xコントロールの実験】

termを起動して，[CTRL]+[X]をキー入力する．[CTRL]+[X]は押すごとにX-ONとX-OFFを交互に送出するので，1度目のキー入力後には相手側からの送信が止まっている状態になっているはずである．実際に相手側からキー入力してみると，こちら側の画面には何も表示されない．こちら側からもう一度[CTRL]+[X]を入力すると，X-ONとなってさきほど相手側でキー入力して送信保留となっていた文字が画面に表示される．Xコントロールはちゃんと動いているようである．

【RS/CS制御の実験】

今度は[CTRL]+[R]をキー入力する．[CTRL]+[R]は，キー入力するたびにRTSをON/OFF交互に行う．[CTRL]+[R]を入力後には，相手側の送信が止まる状態になっているはずである．実際に相手側からキー入力してみると……こちら側の画面に文字が出てくる．出ないはずなのだ．それどころか，[CTRL]+[R]を入力した時点でこちら側からの送信ができなくなってしまった．プログラムにまちがいがあるとは思えないので（いい切ってしまうところが恐ろしいところである），結線を疑ってみることにする．

コラム1で書いたようにRS/CS制御は専用の信号線を使うので，こちら側のRTSが相手側のCTSに，こちら側のCTSが相手側のRTSに接続されていなければならないのだが，現在の配線はそうなっていないではないか．ハードウェアに詳しい知人の話などを総合して，ようやく事情がのみこめてきたのだが，さきほど書いた配線図は「正しいRS-232Cのクロス結線」であり，98同士をつなぐクロスケーブルはこちら側のCTSが相手側のRTSに接続されるように図13のような結線をしなければならないとのことである．

またハンダ付けのやり直しである．やれやれ．

気をとりなおして，ケーブルの結線を図13のように変え，再度実験を試みる．[CTRL]+[R]をキー入力するたびに，相手側からの送信が止まったり再開したりする．ケーブルの結線しだいでRS/CS制御がうまくいったりいかなかったりするのを体験したのだからと，さきほどの苦労を自らなぐさめる筆者であった．

ところで，これはケーブル自作の後

パソコン通信用語集
ビジートーン　話中音ともいい，電話で相手が話中だったときの発信音．モデムはこの発信音で回線の接続状態を知ることができる．

日談になるが，パソコンショップでおもしろいものを見つけた．「RS-232Cミニワイヤリングアダプタ」というものである．RS-232Cコネクタよりちょっと大きめくらいで，片方がオスのコネクタ，もう片方がメスのコネクタになっている．コネクタの上面には本体の両端についているオスメスの各ピンにつながっているピンが剣山のように出ていて，ここを金具のついたリード線で自由に結線できるようになっている（写真1）．

COLUMN

バイナリファイルの送受
XMODEMの通信手順

テキストファイルを送る場合は，ファイルの末尾を示す制御コードとしてEOF（End Of File = 0x1A）を利用することができるが，バイナリファイルの場合はデータ中にすべてのコードが使われる可能性があるために制御コードとデータの区別ができなくなるおそれがあるので制御コードによる処理をすることができない．そこで，「どこからどこまでがデータなのか？」を双方であらかじめ決めてから送受する必要が出てくる．

また，バイナリファイルはプログラムやデータなどを扱うので，テキストファイルとくらべて内容を厳密に送受できないと不都合が生じる関係で，受信したデータのチェック機能も欲しいところである．こうした必要性から何種類かのバイナリデータ送受用手順が利用されている．このなかで，大半のネットワークでサポートされており，手順も簡単なXMODEMを例にしてバイナリ送受の手順を説明してみよう．

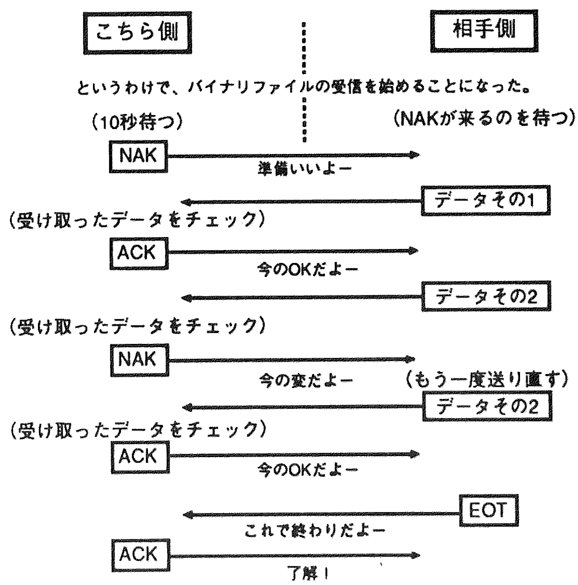
前述したように，バイナリデータはデータ中に制御コードを入れることはできないので，どこで始まりどこで終るかをデータ本体と別に取り決める必要がある．XMODEM手順ではこの問題を固定長ブロック形式を用いることで解決している．XMODEMでのデータ通信は常に132バイト（このうち，データの正味は128バイト）の固定長ブロックでデータを送受する．また，このブロックにはブロックの正当性をチェックするためのブロック番号やデータの内容チェックのためのチェックサムなどが付加されている．また，ブロックの送受信をするタイミングを取るために，1バイトの制御コードを別にやり取りしている．XMODEMでのデータブロックの送受の手順を図Aに示す．

「ACK」，「NAK」，「EOT」というのが，1バイトの制御コードである．これらのコードでタイミングを取っておいて，データブロックを1つずつ送受する．データブロックは図Bのような構造である．

具体的なXMODEM送受信プログラムの処理手順をまとめてみる．

●XMODEM受信プログラムの処理手順

①受信データを書き込むファイルをオープンして，10秒間待つ．
②NAK（0x15）を1バイト送信する．
③データ受信待ち*
④1バイト受信したら以下のチェックを行う．
・データがEOT（0x04）だったら⑨へ（受信終了）
・データがSOH（0x01）だったら⑤へ（データ受信開始）
・それ以外だったら②へ戻る
⑤131バイト分のデータ受信待ち*
⑥以下のデータチェックを行う．
・ブロック番号は妥当か？（前回＋1または255から0になっているか）
・255－ブロック番号の値がブロック番号と一致するか？
・データ本体のバイト毎の総和の下位バイトがチェックサムと一致するか？
すべてのチェックがOKならば⑦へ
1つでもNGだったら②へ戻る．ただし，同じブロックが連続して10回NGになった場合は，受信エラーとして処理を強制中断する．
⑦データ本体をファイルに書き込む．



図A XMODEMでファイルを受信する場合

パソコン通信用語集
フォールバック　上位プロトコルから下位プロトコルに移行すること．相手のモデムより自分のモデムが上位プロトコルをサポートしているとき起こる．

これとストレートのケーブルが1本あれば，リード線をつなぎ変えることによって，どんな結線のケーブルでも簡単に作ることができるのだ．ハンダ付けをやりなおす必要もない．今回のような実験が気軽にできるようになるのはもちろんだが，何種類ものRS-232Cケーブルを買うのであればもっていて損はないかもしれない．

⑧ACK（0x06）を1バイト送信して③へ戻る

⑨ACK（0x06）を1バイト送信して，ファイルをクローズして終了．

＊：データ受信の際は，1バイト受信しようとするたびに10秒間のタイムアウトを設定する．10秒経過しても受信データがない場合には，NAKを1バイト送信してからもう一度10秒以内の受信待ちとする．これを10回繰り返してもデータが受信できないときは，通信エラーとして処理を強制中断すること．

●XMODEM送信プログラムの処理手順

①送信データを読み出すファイルをオープンする．

②NAK（0x15）の受信を60秒間待つ．60秒経過してもNAKが受信できないときは，処理を強制中断する．

③送信データをファイルから読み出し以下の手順でブロックを作成する．

・送るべきデータが残っていなければ⑦へ（データ終了）

・先頭にSOHを付加する．

・ブロック番号を付加する．

・255－ブロック番号を計算して付加する．

・データを128バイト付加する．残っていたデータ量が128バイトに満たない場合は，1Ahで埋めて128バイトにする．

・128バイトのデータのバイトごとの総和を計算し，結果の下位1バイトを付加する．

④できあがった132バイトのデータを送信する．

⑤受信データを10秒間待つ．

⑥1バイト受信したら以下のチェックを行う．

・データがACKだったら③へ（次のブロック送信）

・データがNAKだったら④へ（データ再送）

・それ以外だったら⑤へ戻る

ただし，10回繰り返してもACK／NAKが受信できない場合は，処理を強制中断する．

⑦EOT（0x04）を1バイト送信する．

⑧受信データを10秒間待つ．

⑨1バイト受信したら以下のチェックを行う．

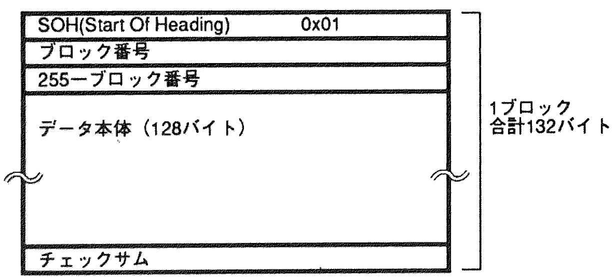
・データがACKだったら⑩へ（送信終了）

・データがNAKだったら⑦へ（EOT再送）

・それ以外だったら⑧へ戻る

ただし，10回繰り返してもACK／NAKが受信できない場合は，処理を強制中断する．

⑩読み出したファイルをクローズして処理終了．



SOH	：ブロックの始まりを示すデータ
ブロック番号	：送信したデータブロックの通し番号 1から始まり、1つずつ増えていく。 255の次は0となり以降はまた+1していく。
255ーブロック番号	：ブロック番号を255から引いた値。 ブロック番号のチェック用に使用する。
データ本体	：128バイトずつ区切って送る。 最終ブロックで128バイトに満たない場合は 足りないぶんを1Ahで埋めて128バイトとする。
チェックサム	：データ本体を1バイトずつ加算していった結果の 下位1バイトをセットする。データ内容のチェックに使用する。

図B XMODEMのデータブロック構造

パソコン通信でもスタンダードなバイナリ転送手段となっていたXMODEMだが，128バイトという細かな単位の送受のため無駄が多くて速度が上がらない，チェックサムのみでは厳密なデータチェックができない，固定長ブロックのために送信側と受信側でファイルサイズが異なる場合が出てくる，前もって何バイトのデータ送受が行われるのかわからないなどの欠点からデータチェック方法や転送単位を改善した拡張XMODEMや大手ネットワークが独自にもっているバイナリ手順などに取って代わられつつある．

しかし，プログラムも容易であるし，バイナリ送受手順の基本ともいえるものであるので，ぜひ一度自作されることをお勧めする．

パソコン通信用語集
フォールバックキャラクタ機能　エラー訂正機能のないモデムをオートリライアブルで接続するときに，特定のキャラクタでエラー訂正機能なしと判断させる機能

いざ，モデムの向こう側の世界へ

さて，コンピュータ同士の通信は開通したようだ．とうとうデータは，コンピュータの匡体からケーブルを伝わって別のパソコンに移動できるようになった．この勢いをかって今度は，電話回線を使って家の外へ飛び出してみよう．

わかりきったことではあるが，パソコンと電話回線をつなぐためには，モデムが必要である．まずはモデムとパソコンの接続だが，モデムに付属のケーブルを使えばよいのでこれは問題ないだろう．モデム用のケーブルは，さきほど説明した「ストレートケーブル」というものである．結線はたいがい図14のようになっているはずである．

パソコンとモデムがつながったら，さきほどのターミナルを起動してみよう．相手がパソコンであろうとモデムであろうと，あるいはたとえケーブルが違おうとも，こちら側からみればRS-232Cを使ってデータの送受を行うことにはかわりないので，同じターミナルソフトが使えるのである．

ターミナルが起動したら，モデムの初期化を行うためのコマンド「atz」を入力してみる．

無事にモデムとの交信ができている場合には，たぶん図15のようになると思う．ここで問題なのは「atz」と入力したはずなのに，画面には「aattzz」と表示されている点である．ゆっくりキーを入力していくと気がつくと思うが，1文字入力したときに同じ文字が画面にもう1つ出てくる．実はあとで表示される文字はモデムからの受信データなのである．からくりは，図16のようになっている．

この場合のように送信したデータを相手側がそのまま返してくるのを「エコーバック」と呼ぶ．多くのネットワークはこのエコーバックをしてくるので，このままにしておくとキーから入力した文字はすべて2つずつ画面に表示されることになる．これでは少々(でもないが）具合が悪いので，こちらのプログラムのほうで，キー入力した文字は画面表示しないという仕様にすることが必要である．

さて，こういった問題はあるものの，とりあえずはモデムとの通信が成立した．やろうと思えばこのまま電話回線を使ってネットワークに接続することができる．そのためにこちらから送信するデータは次の2つに大別できる．

- **モデムへのコマンド（初期化や電話番号など）**
- **ネットワークへのコマンド**

どちらにしても，こちら側からみればRS-232Cを使ってコマンドを送出することにはかわりない．モデムやネッ

図14

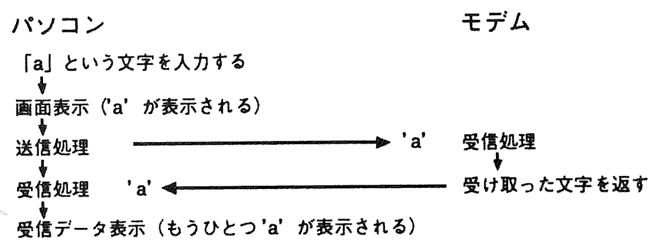
1 GND	1 GND
7 GND	7 GND
2 TxD	2 TxD
3 RxD	3 RxD
4 RTS	4 RTS
5 CTS	5 CTS
6 DSR	6 DSR
8 DCD	8 DCD
20 DTR	20 DTR

図15

```
>>term
aattzz                    ←キーボードから「atz」と入力する

OK                        ←モデムからの応答
```

図16



パソコン通信用語集
物理伝送速度　実際にデータを伝送している速度．1秒間に何ビットのデータが送れるかで表現する．2400bpsなら1秒間に2400ビットのデータを送受信することができる．

リスト16 APLIB.S

```
; ==============================================================
; APLIB.S
;
; Automatic pilot libraly for SLEAP system
;
; Create : 1991.10.12
;
;
; Copyright 1991 By Justech Labo. (H.kanoshima)
;
; ==============================================================

; ==============================================================
var     logfilep
; ==============================================================

buffer  rbuffer 2048
var     rbufferp

[ init_rbuffer  ; (none) -> (none)
        0 rbufferp ! ]

[ next_rbufferp ; rbufferp -> rbufferp+1
        1+ dup 2048 =
        if      drop 0
        then ]

[ prev_rbufferp ; rbufferp -> rbufferp-1
        ?dup
        if      1-
        else    2047
        then ]

[ rbuffer_ptr   ; rbufferp -> rbuffer_ptr
        rbuffer + ]

[ put_rbuffer   ; char -> (none)
        rbufferp @ rbuffer_ptr b!
        rbufferp @ next_rbufferp rbufferp ! ]

[ tailrchr      ; (none) -> char
        rbufferp @ prev_rbufferp rbuffer_ptr b@ ]

; ==================================================================

[ file_cprint   ; char -> (none)
        dup $0d =
        if      drop
        else    logfilep @ fcprint drop
        then ]

[ tprint        ; char -> (none)
        ?dup
        if      dup cprint
                dup file_cprint
                put_rbuffer
        then ]

; ==================================================================

[ log_open      ; string -> (none)
        "a" fopen logfilep !
        logfilep @ 0 =
        if      "Can't open logfile¥n" sprint 1 exit
        then ]

[ log_close     ; (none) -> (none)
        logfilep @ fclose drop ]

; ==================================================================

[ read_rbuffer  ; address , size -> (none)
        >r                      ; save size
        rbufferp @
        i
        do
                prev_rbufferp
        loop
        swap                    ; top:buffer , 2nd : rbufferp
        r>
        do
                over rbuffer_ptr b@ over b!
                1+ swap next_rbufferp swap
        loop
        0 swap b!               ; terminate string
        drop ]

buffer  chkstr  160

[ tailchr       ; string -> char
        dup slen + 1- b@ ]

[ exist_string  ; string -> 1/0
        dup tailchr tailrchr =
        -if     drop 0 return
        then

        dup slen chkstr swap read_rbuffer
        chkstr s= ]

[ wait_string   ; string -> 1
        begin
                rs_receive_data_wait tprint
                dup exist_string
                if      drop 1 return
                then
        repeat ]

[ wait_2string  ; string , string -> (none)
        begin
                rs_receive_data_wait tprint
                over exist_string
                if      2drop 1 return
                then
                dup exist_string
                if      2drop 2 return
                then
        repeat ]

[ wait_3string  ; string , string , string -> (none)
        begin
                rs_receive_data_wait tprint
                2 pick exist_string
                if      3drop 1 return
                then
                1 pick exist_string
                if      3drop 2 return
                then
                0 pick exist_string
                if      3drop 3 return
                then
        repeat ]

[ wait_4string  ; string , string , string , string -> (none)
        begin
                rs_receive_data_wait tprint
                3 pick exist_string
                if      4drop 1 return
                then
                2 pick exist_string
                if      4drop 2 return
                then
                1 pick exist_string
                if      4drop 3 return
                then
                0 pick exist_string
                if      4drop 4 return
                then
        repeat ]

; ==================================================================

[ wait_and_print_10msec ; time(10msec) -> (none)
        rs_set_timer
        begin
                ?rsinput
                if      rs_receive_data tprint
                then
                ?cinput
                if      1 exit
                then
                rs_chk_timer
                if      return
                then
        repeat ]

[ wait_10msec   ; time(10msec) -> (none)
        rs_set_timer
        begin
                ?cinput
                if      1 exit
                then
                rs_chk_timer
                if      return
                then
        repeat ]

[ init_aplib
        "sleap.log" log_open
        init_rbuffer ]

[ finish_aplib
        log_close ]

"Compiled -aplib.s-¥n" sprint
```

トワークに必要なコマンドを順々に送り出してやればいいだけの話である．

通信ソフトの華【オートパイロット】

せっかく通信ソフトを作ってネットワークにアクセスするのだから,「さすがは通信ソフト」と呼ばれるような大向こう受けする機能を追加してみよう．ネットワーカなら必ず一度は耳にしているであろう「オートパイロット」である．

オートパイロットとは，文字どおり「自動運転」のことである．通常ならネットワークに接続したら，人間が画面に表示されたメッセージを読んだり，キーボードからコマンドを入力したりする．ネットワークも慣れてくると次第に操作するコマンドは決まってきて，いわゆる「巡回経路」もできてくる．こうなってくるとモノグサはどこにでもいるもので（筆者もその１人である)，この手順を何とか自動化できないものかと考える．そこでオートパイロットの登場である．オートパイロットはあらかじめ組み込まれた手順どおりに自動的にネットワークとの交信を行う．

基本的には，ネットワークに接続して，ホストがコマンド待ちになったところで，あるコマンドを自動送信する．ホストがそのコマンドを受け付けると，なんらかの反応が返ってきてまたコマンド待ちになる．その状態を検知して，また次のコマンドを自動送信する．この繰り返しである．

表３　APLIB.S に含まれる単語

init_aplib	入力：なし 出力：なし	受信データ記録ファイルをオープンする．ファイル名は「SLEAP.LOG」としている． ファイル名を変更したい場合は，この単語を変更する．このファイルは，常にカレントディレクトリに作成する．すでに同名のファイルが存在するときは，そのファイルの末尾に追加する形で記録を続ける．したがって，そのままにしておくと通信のたびに果てしなく記録ファイルが大きくなりつづけることになるので，注意してほしい． また，この単語の中でオートパイロットに使用する受信データ保存バッファの初期化も行う． なお，この単語では回線やモデムの初期設定などは行わない．これらをふくんだ一連の初期化処理の実例は，このあと実際にネットワークへ接続するためのプログラムで紹介するので，参考にしてほしい．
finish_aplib	入力：なし 出力：なし	受信データ記録ファイルをクローズするだけである．RS-232C のための後処理などは，後述のネットワーク用プログラムを参照のこと．
wait_string	入力：文字列をスタックに積む 出力：１がスタックに積まれる	スタックに積まれた文字列を待ちながら受信を続ける．受信データは画面と記録ファイル両方に書き込んでいく．指定の文字列を受け取った時点でスタックに１を積んで処理を終了する．
wait_2string	入力：文字列を２つスタックに積む 出力：なし	基本的な動作は「wait_string」と同じであるが，この単語はスタックから２つの文字列を受け取って，「これかこれを受信したら……」という監視を行う．指定文字列のうちいずれか片方を受信した時点で処理を終了する．ネットワークの場合，状況に応じて複数のプロンプトのいずれかが返ってくるということはよくあるので，この単語はかなり役にたつのではないかと思う．
wait_3string	入力：文字列を３つスタックに積む 出力：なし	基本的な動作は「wait_string」と同じであるが，この単語はスタックから３つの文字列を受け取って，「これかこれかこれを受信したら……」という監視を行う．指定文字列のうちいずれか１つを受信した時点で処理を終了する．
file_cprint	入力：１文字をスタックに積む 出力：なし	スタックに積んだ文字を記録ファイルに書き込む． 通常，受信データなどは後述する「tprint」によって，画面とファイル両方に書き込むが，とくにファイルのみに何かデータを書き込みたいときはこの単語を使う．
tprint	入力：１文字をスタックに積む 出力：なし	スタックに積んだ文字を ・画面 ・記録ファイル ・文字列監視用バッファ にそれぞれ書き込む． 受信データの表示には「cprint」ではなく，こちらの単語を使うこと．

パソコン通信用語集
フロー制御　受信データが端末の処理速度より速いとき，モデムのバッファがいっぱいになったときなどにデータが消失しないよう端末またはモデムに対して行う制御．

こういうふうに書いてしまうと「華」のありがたみが薄れてしまいそうだが，結局のところ，オートパイロットを実現するために必要なのは，「こういう場合にこうしろ」というプログラムである．これをもっと具体的にいうと，機能として必要なのは

①コマンド文字列を送信する．
②ある応答文字列（これは複数のうちのいずれかかもしれない）を受信するのを待つ．
③応答によって，次のコマンド文字列を送信する．

ということである．人間がネットワークを利用するときの操作をこのようにプログラム化していけばいいわけだ．問題は，そのプログラム化がたやすい環境があるかどうかである．

というわけで，SLEAPのためのオートパイロット用単語集「APLIB.S」を作成することになった．コマンド文字列の送信はターミナルの時に作った「COMLIB.S」を使えばいいので，乱暴にいってしまえば②を実現するだけでオートパイロットの機能は実現できることになる．

1つ大切なことを忘れていた．オートパイロットは文字どおりの自動運転であるから，プログラムを起動したあとは食事に行ってもいいし，寝てしまってもいいはずなのだが，その間の受信データを画面表示しているだけでは，そうはいかない．自動運転で表示されている通信データをパソコンの前でじっと見ていなくてはならないのでは，何のためのオートパイロットかわからない．

COLUMN

マクロプログラマ
マクロは守備範囲が広く，手軽に結果を得られる

世間にはプログラムは書けないがマクロなら書ける「マクロプログラマ」なる人種が存在する．ご本人は「いやぁ，プログラムなんてとてもとても．マクロなら少しは書けるんですが」などと恥ずかしそうにおっしゃる．

しかしマクロもまた言語であり，その言語を使って書かれたものはプログラムである．実に不思議な話なのだが，彼らはいわゆる汎用のプログラミング言語を使うことはしないのである．思うにこれはこういうことなのであろう．

汎用のプログラミング言語というのは汎用であるがゆえに，1つ1つの命令あるいは機能の守備範囲が狭い．プログラマはその小さな命令を組み合わせ，縦横無尽に操って必要な動作を実現する．あまり複雑な動作やサービスはかえって迷惑なのである．

たとえばたった1つの命令で，「ファイルを読んでそれをウインドウ内に表示し，カーソルキーで自由にスクロールさせる」ということが実現できるとしよう．これはたいへん便利ではあるが，たとえばウインドウの色を変えたいとかキー操作を変更したいというような要求には応じてくれない．

またたとえば，存在しないファイルを読もうとしたら，その旨を表示してプログラムを終了させてくれるという命令があるとしよう．どのみちファイルは読めないのだから終了させる手間がなくてありがたいようなものだが，この命令を使う限りもう一度ユーザーに正しいファイルを指定させるというサービスは実現できない．

そこでプログラマは仮にこういう命令があっても使わず，そのかわり単にファイルがあるかないかを調べる命令とか，指定した色で画面に線を引くだけの命令を使って必要な機能を実現するのである．

ところが，そういうことを職業にしているかあるいはそれが楽しくてしょうがないという困った人は少数派だ．おおかたの人は本業があり，それに役立たせるためにやむをえずプログラムを書くのである．趣味でプログラムを書く人でも，早く結果を見たい，プログラミングに十分な時間をかける余裕がないということは多いだろう．

そういう意味では，1つ1つの命令の守備範囲が広いマクロはありがたい存在である．データ処理をするのに専用のデータ処理プログラムを書くのではなく，データベースのマクロを使ってとりあえず満足できる結果を得る．通信ソフトをゼロから書くのではなく，通信ソフトのマクロを使って，本来それがもっていない必要な機能を実現するのである．

こうした人たちにとってのマクロの優位性はもう1つある．たいていのマクロは，それを書いただけですぐに実行してみることができる．いわゆる汎用のプログラミング言語のようなコンパイルとかリンクといった手順を必要としない．慣れてしまえばなんのことはないのだが，結果とはなんの関係もない手続きを覚えるのはおっくうなものである．

マクロプログラマはこのように，手軽にある程度満足のできる結果を得るためにマクロを好んで使うのであろう．決して汎用言語でのプログラミングができないのではないのだ．彼らはたぶんその気になれば，汎用のプログラミング言語を使って自分の必要とするプログラムを書くことができる．いまマクロでやっていることをもう何レベルか掘り下げるだけでいいのだから．

「マクロくらいしかわかりませんから」などと恥ずかしそうにする必要はまったくないのだ．

パソコン通信用語集
プロトコル（伝送制御手順）　データの伝送を行ううえでの同期の確立，データの形式，誤り制御などを決めた手続き．

そこで，受信データ記録をファイルに書き込む機能が必要になる．通信記録をすべてファイルに書き込んでおいて，あとでゆっくりと眺めるというのはオートパイロットならではの楽しみである．ここに凝りだすと，「通信記録ファイルの整理術」となってそれだけでたいへんな分量になってしまうのだが，APLIB.Sではごく基本的な機能として，起動されてからプログラムが終了するまでのすべての表示文字を同時にファイルに書き込むということにした．オートパイロット用単語集「APLIB.S」の単語は表3のようなものである．

NIFTYにアクセスする

さて，長いことお付き合いいただいてきたが，そろそろ最後の仕上げをする段階にきたようだ．いよいよままで作ってきた通信ライブラリ，コンパイラ，基本単語，通信用単語集を使って，ネットワークへのアクセスプログラムを作る．断言してもよいが，この段階までくると，あとはほとんど自分のやりたいことをならべて書いていくだけでアクセスプログラムを書くことができるはずである．少なくとも，これから紹介するNIFTY-Serve用のアクセスプログラムを見て，何をしようとしているのかは読めばわかると思う．あとは，読者のお好みに合わせて改造や機能追加をしていただきたい．

今回はNIFTY-Serve用のアクセスプログラムを作ってみたが，これは

リスト17 NIFTY.S

```
; ==============================================================
; NIFTY.S
;
; NIFTY-Serve access libraly for SLEAP system
;
; Create : 1991.10.12
;
;
; Copyright 1991 By Justech Labo. (H.kanoshima)
;
; ==============================================================

compile         "stdlib.s"
compile         "comlib.s"
compile         "aplib.s"

[ id                    "xxxxxxxx¥r" ]
[ password              "xxxxxxxx¥r" ]

[ modem_command1        "ATZ¥r" ]
[ modem_command2        "ATE1V1X4¥¥N3¥¥Q4¥r" ]

[ host_number           "ATDTxxx-xxxx¥r" ]

[ send_id               id rs_send_string ]
[ send_password         password rs_send_string ]

[ wait_id_prompt        "-->" wait_string drop ]

[ wait_prompt_nif
        "¥n>" wait_string drop ]

[ wait_prompt_forum
        begin
                "¥nmore>" "続き (改行で表示  E:終了)¥r¥n>" "¥nFORUM>" "¥n>"
                wait_4string
                1 =
                if      "¥r" rs_send_string
                else    return
                then
        repeat ]

[ wait_big_prompt
        begin
                "¥nmore>" "続き (改行で表示  E:終了)¥r¥n>" "¥n>" wait_3string
                1 =
                if      "¥r" rs_send_string
                else    return
                then
        repeat ]

[ wait_prompt_mes       wait_big_prompt ]

[ connect_road2
        200 wait_and_print_10msec
        "@P¥r"          rs_send_string
        "¥n*"           wait_string drop
        "C NIF¥r"       rs_send_string ]

[ connect_nifty
        connect_road2
        wait_id_prompt
        "SVC¥r" rs_send_string
        wait_id_prompt
        send_id
        wait_id_prompt
        send_password ]

[ bye
        "off¥r" rs_send_string
        "¥n*" wait_string drop
        "off¥r" rs_send_string ]

[ init_comport
        set_default_comspec
        rs_initialize ]

[ init_modem
        modem_command1 rs_send_string
        100 wait_and_print_10msec
        modem_command2 rs_send_string
        100 wait_and_print_10msec ]

[ init_all
        init_comport
        init_aplib
        init_modem ]

[ finish_all
        finish_aplib
        rs_finish ]

[ dialing
        host_number rs_send_string
        "CONNECT" wait_string drop ]

[ go_forum      ; string -> (none)
        "go " rs_send_string
        rs_send_string
        rs_send_cr
        wait_prompt_forum ]

[ enter_mes
        "mes¥r" rs_send_string wait_prompt_mes ]

[ mread_all
        "mread¥r" rs_send_string
        wait_big_prompt ]

[ read_mes
        enter_mes
        mread_all ]

[ go_forum_and_read_mes ; string -> (none)
        go_forum
        read_mes ]

[ main
        init_all
        dialing
        connect_nifty
        wait_prompt_nif
        ;==========================
        "fgalbt" go_forum_and_read_mes
        ;==========================
        bye
        finish_all ]

"Compiled -nifty.s-¥n" sprint
end main
```


パソコン通信用語集

文字化け　アナログ信号である音声を送るための電話回線にはノイズが発生する，このため信号が変化し正しいコードが送れないことがある．これを文字化けという．

筆者が加入している大手ネットワークがたまたまここだけというだけの事情で他意はない．ほかのネットのアクセスプログラムも，これをモデファイすることによって簡単に作ることができるはずである．モデファイが簡単だと力説しているだけでは説得力がないので，まずはこのアクセスプログラムのメイン単語を見ていただくことにしよう（リスト18）．

やっていることは，これで全部である．このうちNIFTY-Serve専用となるのは，3:〜8:までである．init_allやdialing，finish_allといった単語は，RS-232Cやモデムの初期化のため，どのネットワークにも共通に使用できる．

ではまず，この部分の単語について解説した表4を掲載する．

次に，NIFTY-Serveに依存した単語を説明する．NIFTY-Serveの応答にも触れておくので，ほかのネットワークへの移植をする場合の参考にしてほしい．

まず，回線が接続されてからIDやパスワードを入力してメニューが出てくるまでのNIFTY-Serveの応答を知っておかなくてはならない（図17：左端の行番号は説明のために付加しているもので，実際の通信時にはない）．

では，順番に接続までの手順をみていこう（表5）．

ここまでの接続手順は，だいたいどこのネットワークもおおすじにおいて同じなのではないだろうか．要は「これがきたら，これを送る」という手順を指示することである．あとは「少し待つ」とか「改行を送るか送らないか」などという微調整になるだろう．

そのほかのNIFTY-Serve専用単語は，表6のようなものである．

NIFTY-ServeのIDをもっている読者なら，ぜひとも実際に接続して動作を体験していただきたい．まず，「NIFTY.S」の中の個人的データを記入する．これらはファイルの冒頭に

リスト18

```
[ main
1:      init_all                    ;RS-232C、モデム及びﾌﾟﾛｸﾞﾗﾑ内部の初期化
2:      dialing                     ;電話をかける
3:      connect_nifty               ;Ｎｉｆｔｙ－Ｓｅｒｖｅへの接続コマンド
4:      wait_prompt_nif             ;コマンド待ち状態になるのを待つ
5:      ;==========================
6:      "fgalbt" go_forum_and_read_mes    ;Ｎｉｆｔｙ－Ｓｅｒｖｅの中でやりたい
7:      ;==========================      ;処理
8:      bye                         ;Ｎｉｆｔｙ－Ｓｅｒｖｅの回線切断コマンド
9:      finish_all ]                ;RS-232Cなどの後処理
```

```
1: CONNECT 2400
2: @P<CR>                 (実際には画面表示されない)
   ^^^^^^
3: FENICS-ROAD 2<CR><LF>
4: HOST NAME?<CR><LF>
5: <LF>*C NIF<CR>
        ^^^^^^^^^
6: <LF>COM<CR>
7: <LF> Enter Connection-ID  --->SVC<CR>
                                 ^^^^^^^
8: <LF> Enter User-ID  --->smple-id<CR>
                           ^^^^^^^^^^^^
9: <LF> Enter Password --->smple-pw<CR>  パスワードは実際には画面表示されない)
                           ^^^^^^^^^^^^
```

図17

表4

●init_allで実行しているのは，次の2つの単語およびAPLIB.Sの中の「init_aplib」である．		
init_comport	入力：なし 出力：なし	set_default_comspecを実行して，RS-232Cの初期化パラメータを設定する．ネットワークへの接続の場合は，1度決めた通信条件を変更することはほとんどないので，常にデフォルトでの初期化でいいだろう．パラメータを設定後，通信ライブラリの初期化ルーチンを呼び出す．
init_modem	入力：なし 出力：なし	モデムへの初期化コマンドを文字列として送出する．ここは所有しているモデムによって多少コマンドが異なることがある．NIFTY.Sの場合は，筆者所有の「NEC COMSTARZ CLUB 24/5」を2400BPS/MNPクラス5で使用するためのコマンドにしてある．コマンドは2回に分けて送出しているが，コマンドとコマンドの間に適当な待ち時間をおいている．このタイミングなどもモデムの機種によって異なる場合がある．いずれにしても，このままのコマンドで動作しない場合はお手もちのモデムのマニュアルなどを参照していただきたい．モデムのコマンドは，NIFTY.Sの冒頭にまとめた個々の環境用の定義文に含まれるので，そこを修正する．
dialing	入力：なし 出力：なし	電話番号の文字列を送出する．この場合の電話番号文字列はいわゆる「ダイアルコマンド」である．プッシュ／ダイアルの回線種別もこの中にふくまれる．送出が終了したら，モデムから回線接続をしめす「CONNECT」という文字列が送信されてくるのを待つ．電話番号は，NIFTY.Sの冒頭にまとめた個々の環境用の定義文に含まれるので，そこを修正する．
finish_all	入力：なし 出力：なし	finish_aplibとrs_finishを呼び出して，内部状態およびRS-232Cの後処理を行う．

パソコン通信用語集
モデムポート　モデムから見て相手モデム側，すなわちモジュラージャックの入出力をさす．

集めてある．

これで準備万端整った．あとは，プログラムを起動するだけである．NIFTY.Sは，プログラムの中に基本単語のコンパイルとmain単語の即実行コマンドを埋めこんであるので起動は簡単である．

```
A>SLEAP NIFTY.S<ret>
```

起動の前に，本体とモデムがきちんと接続されていて，モデムの電源がONになっていることを確認してほしい．起動したプログラムは，モデムを初期化して電話をかけ，NIFTY-Serveに接続してフォーラムの電子会議室を読んで回線を切る．人間は黙ってみているか，その場を離れて食事・睡眠・仕事などお好み次第である．その間の記録はすべてカレントディレクトリに「SLEAP.LOG」というファイル名で残っている．

NIFTY-Serveに接続して，何をするかはマクロプログラムを書く人間の指図によるが，サンプルでは「ソフトウェアギャラリー・ベストツール（FGALBT）」というフォーラムに入って，すべての未読発言を読むという作業をさせている．

図18はその記録ファイルの内容である．

まとめ

「通信ソフトの作り方」，いかがだっただろうか．何度も書いてきたような気もするが，制御用LSIや割り込みのタイミングなどがパーツ化されていれば，基本的な通信ソフトというものも，そう特別にむずかしいものではない．その意味で，今回ご紹介した通信ライブラリを活用していただければ幸いである．

また，実際にネットワークなどの通信機能を利用されている読者には，

表5

connect_nifty	入力：なし 出力：なし	回線が接続してから，パスワードを入れてNIFTY-Serveのメニューが出てくるまでの一連の処理を行う．処理手順は以下のとおり． ●connect_road2 「CONNECT 2400」を受信したときに実行がこの単語に移ってくる．NIFTY-Serveからの注意として，回線が接続してから5秒程度待って次のコマンドを送信するようにということであるのだが，せっかちな筆者は2秒待つことにしてある．これで数百回の接続を行っているが，今のところ不具合は出ていない．2秒待ったら，2:の「@P<CR>」コマンドを送信する．次は5:の「C NIF<CR>」を送出するために，「<LF>＊」という受信データがくるのを待つ．「C NIF<CR>」を送出したらこの単語の処理は終了である． ●wait_id_prompt NIFTY-Serveのサービスを開始するためには，「サービス種別」，「ID」，「パスワード」の3つの入力をしなければならない．それぞれの入力タイミングで「-->」というプロンプトを送信してくるので，これを待つための単語が「wait_id_prompt」である． ●"SVC¥r" rs_send_string サービス種別は，通常の接続の場合常に「SVC」である． ●wait_id_prompt 再び，「-->」というプロンプトを待つ． ●send_id 自分のIDを送信する． ●wait_id_prompt 再び，「-->」というプロンプトを待つ． ●send_password 自分のパスワードを送信する．

パソコン通信用語集
リザルトコード　モデムへの操作コマンドに対し，その結果を端末に返すコード．ATコマンドの場合キャラクタタイプとディジット（数字）タイプがある．

SLEAPのマクロプログラム機能を実用に役立てていただきたい．機能満載の高機能通信ソフトも魅力ではあるが，たとえば電子メールを受け取るだけとか，あそこにあるあのソフトをダウンロードするだけとかいう専用ソフトはいかがだろうか？　用途に合わせて通信ソフトを使い分けてみるのも通信の楽しみの1つと思うのだが．そしてまったく通信をしたことのない読者の方には，ぜひとも一度何らかの形で体験されることをお勧めしたい．RS-232Cのケーブルを通して，もしかしたらあなたの人生観を変えてしまうようなできごとが待っているかもしれない．

今回は特集ということで，通信ライブラリから始まって，とうとうコンパイラを1本作りあげることになったが，書くほうもさまざまな面で勉強させていただいた．この機会を与えてくださった編集部の方に感謝する次第である．そして，この長丁場を最後までお付き合いいただいた読者の方には心からお礼申し上げる．

表6

wait_prompt_nif	入力：なし 出力：なし	NIFTY-Serveのメインプロンプトは「>」である． このプロンプトが送信されてきたらコマンド待ちになったと判定して処理を進める．
go_forum	入力：フォーラム名文字列をスタックに積む 出力：なし	NIFTY-Serveの代表的なサービスに「フォーラム」というのがある．フォーラム自体の説明は，NIFTY-Serveのマニュアルや紹介記事に任せるとして，メインメニューからフォーラムのサービスを開始するためのコマンド手順だが， >GO フォーラム名<CR> となっている．この単語では受け取ったフォーラム名に「GO」と改行を付けて送出し，フォーラムサービスが開始してコマンド待ちになるまで待つ．
wait_prompt_forum	入力：なし 出力：なし	フォーラムサービスを利用しているときに，コマンド待ちの状態になる(直前のコマンドが終了したという意味もある)のを待つ単語である．フォーラムサービス中には次のいずれかのプロンプトを送信してくる．この単語は，通常の受信処理を続けていって，どちらかのプロンプトを受信したら処理を終了する． <LF>FORUM> <LF>> 実はもう1つ「<LF>more>」というプロンプトがある．このプロンプトは画面のページ制御を行っている場合に，指定行数で一時送信を停止したときのもので，こちらからCRコードを送信すると送信が続行される．この単語の中でも「<LF>more>」を受信したときには，CRコードを送って応答することにしてある．
enter_mes	入力：なし 出力：なし	NIFTY-Serveの各フォーラムサービスは「電子会議」「掲示板」「データライブラリ」などのフォーラムメニューからなっている．この単語はその中の「電子会議」を選択するためのものである．
mread_all	入力：なし 出力：なし	この単語は，フォーラムサービス中の「電子会議」を利用しているときに，「未読の全発言を表示する」というコマンド「MREAD」を送出し，表示が終了してプロンプト状態になるのを待つ単語である．
bye	入力：なし 出力：なし	NIFTY-Serveのサービスを終了する単語である．NIFTY-Serveの切断コマンドは以下の手順で入力する． >bye<CR> LOG IN --- 91/10/16 16:14:47 LOG OUT --- 91/10/16 16:15:15 ご利用時間は，　00分28秒でした． ご利用誠にありがとうございました． Clear PAD Host requested clearing the call HOST NAME? <LF>＊off<CR> この手順に従って，まず「off<CR>」を送信し，「<LF>＊」というプロンプトの受信まで待って，もう一度「off<CR>」を送信する．

パソコン通信用語集
リトレイン　データ通信中に回線上のノイズなどの影響で正常な通信ができないときに，キャリアの送出を中断し再接続を行うこと．

リスト19

```
[ id                    "xxxxxxxx¥r" ]
●xxxxxxxxの部分に自分のＩＤを記入する。

[ password              "xxxxxxxx¥r" ]
●xxxxxxxxの部分に自分のパスワードを記入する。

[ modem_command1        "ATZ¥r" ]
●ここはおそらく変更の必要がないだろう。

[ modem_command2        "ATE1V1X4¥¥N3¥¥Q4¥r" ]
●接続がうまくいかなかったら、使用するモデムに合わせて修正する。

[ host_number           "ATDTxxx-xxxx¥r" ]
●xxx-xxxxの部分に接続するアクセスポイントの電話番号を記入する（市外局番が必要な場合は当然そこ
から）。また、上記の例はプッシュ回線用なので、ダイアル回線の場合は「ATDPxxx-xxxx¥r」となる。
```

```
ATZ
OK
ATE1V1X4¥N3¥Q4
OK
ATDTxxx-xxxx
CONNECT 2400

FENICS-ROAD 2

HOST NAME?
*C NIF

COM
 Enter Connection-ID  --->SVC
 Enter User-ID  --->sampl-id
 Enter Password --->

ようこそNIFTY-Serveへ
  Copyright (C) 1991
 by NIFTY Corporation
 All Rights Reserved

前回LOG OUT 91/10/16    15:28:36

NIFTY-Serve     TOP
 1. サービス案内                     2. 電子メール
 3. 掲示板                           4. CBシミュレーター
 5. フォーラム                       6. ニュース／スポーツ／天気予報
 7. ビジネス／科学／技術             8. 教育／就職
 9. ショッピング／アドコーナー      10. 旅行／イベント
11. 趣味／ライフスタイル／ゲーム    12. ワープロ／コンピュータ
13. 海外データベース(INFOCUE)       14. CompuServe
15. ビギナーズコーナー               E. 終了
>GO FGALBT

    ----------------------------------------------------------------
    最高のパソコン使用環境を提供するＦＧＡＬベストツールへようこそ！
    ----------------------------------------------------------------

  ※※※※※※※※※※  中略  ※※※※※※※※※※※※

<FGAL・ベストツール>    FGALBT
FORUM>mes
番号  発言 (未読)   最新    会議室名

  ※※※※※※※※※※  中略  ※※※※※※※※※※※※

>mread
- FGALBT  MES( 7):■母：天衣無縫：      メニュー関連ソフト 91/10/16 -

  ※※※※※※※※※※  中略  ※※※※※※※※※※※※

>off

        LOG IN  ―――  91/10/16    16:14:47
        LOG OUT ―――  91/10/16    16:15:15

  ご利用時間は、  00分28秒でした。
  ご利用誠にありがとうございました。

Clear PAD
Host requested clearing the call

HOST NAME?
```

図18

パソコン通信用語集
リライアブルオンリー　通信開始時に相手のモデムがV.42またはMNPなどエラー訂正機能付きのみ接続し，それ以外は回線を切断する機能。

データ通信プログラミングの参考のために

データ通信に関する書籍には，コンピュータ間通信一般について論じたもの，その中でもとくに通信プロトコルについて解説したもの，RS-232CやGP-IBなどのインターフェースをプログラムからどのように制御するかという具体的なテクニックを記述したものなどがある．パソコン側のインターフェースを制御する知識さえあれば，プログラムは作れるが，1985年の通信の自由化以来，データ通信の普及はめざましく，ISDNなどの高速なデジタル通信が家庭でも利用できるようになってきた現在，プログラマたるもの，データ通信に関してある程度の技術的な知識は身につけておきたいものだ．

データ伝送技術実用ノウハウのすべて



宮崎誠一著
CQ 出版社
2,500円

一般的なローカルのデータ伝送のハードウェアとハードウェアに直結したソフトウェアの設計がひととおりできるようになることを目的とした実用書．「第1部　データ伝送の基礎技術」，「第2部　データ伝送固有の技術」，「第3部　伝送高度化の技術」の3部構成．

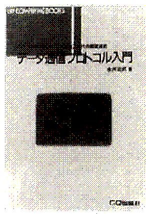
データ通信技術セミナー



宮崎誠一著
CQ 出版社
1,650円

パソコン通信のように，比較的簡易なデータ通信で利用する通信手段について，その仕組みを解説した本．通信回線の種類とその仕組み，RS-232C，RS-422などのインターフェース，通信プロトコル，パソコン通信，VAN，OSIなどの広い範囲のテーマが取り上げられている．

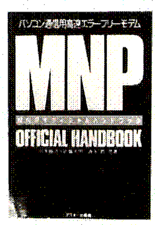
データ通信プロトコル入門



永井正武著
CQ 出版社
2,000円

パソコン通信のための情報通信プロトコルの入門書．前半4章は，情報通信プロトコルの基礎として，データ通信の基礎やOSI参照モデル，伝送制御手順，NCUと自動発着信方式などについての解説．後半では無手順，ベーシック手順，非同期系手順，BSC手順，JCA手順，全銀プロトコル，BTAM，XMODEM，YMODEM，KERMITなどのプロトコルについての解説．チェックサムやCRCチェック値の計算関数のサンプルなども記載されている．

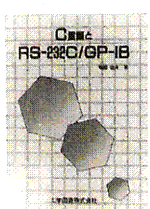
MNPオフィシャルハンドブック



山本勝之・佐藤英明・森下哲共著
アスキー出版局
1,850円

パソコン通信で用いられる無手順通信で，エラー訂正を行うための通信方式である，MNP（Microcom Networking Protocol）についての解説書．MNPモデムのエラー訂正のしくみ，MNPの各クラス別の機能の概要，MNPの各機能を使うためのATコマンドの解説など，MNPモデムに関する理解を深めたい人向けの1冊．

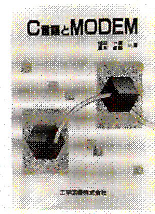
C言語とRS-232C/GP-IB



磯部俊夫著
工学図書
2,060円

C言語を使って，RS-232C，GP-IBインターフェースを利用するプログラムを書くための解説書．RS-232Cインターフェースについては，μPD8251制御をはじめ，各種の関数サンプル，簡単なターミナル機能をもつプログラムやファイル転送プログラムが掲載されている．

C言語とMODEM



植木一廣・黒木直樹共著
工学図書
1,500円

Lattice C用の通信用ライブラリ「C-MODEM98」を使って，NEC PC-9800シリーズ用の簡易ターミナルプログラムを作成する手順を解説する．パソコン通信のための基礎的な知識や，データ伝送方法の概要，ヘイズATコマンドの概要などについても触れている．

トラ技コンピュータ 1991年10月号 特集 データ通信／転送プログラミング入門



CQ 出版社
650円

RS-232C，モデム，ISDNボードなどを活用するためのノウハウを集めた特集．内容は「RS-232Cによるデータ転送の基礎とプログラミング」「モデムを使った通信プログラムの作り方」「ファイル転送ユーティリティの作り方」「図形データ転送ユーティリティの作り方」「PC-9801用RS-232C拡張ボードの使い方」「PC-9801とISDNボードによるデータ通信」．

パソコン通信用語集

論理伝送速度　2400bps以上のモデムは一般にエラー訂正やデータ圧縮が行われるため物理速度以上の速度が得られる．これを論理伝送速度という．

パソコン博物館 XIV

カラクリ伝統のパソコン

PASOPIA 7

東芝パソピア7

井戸禎光

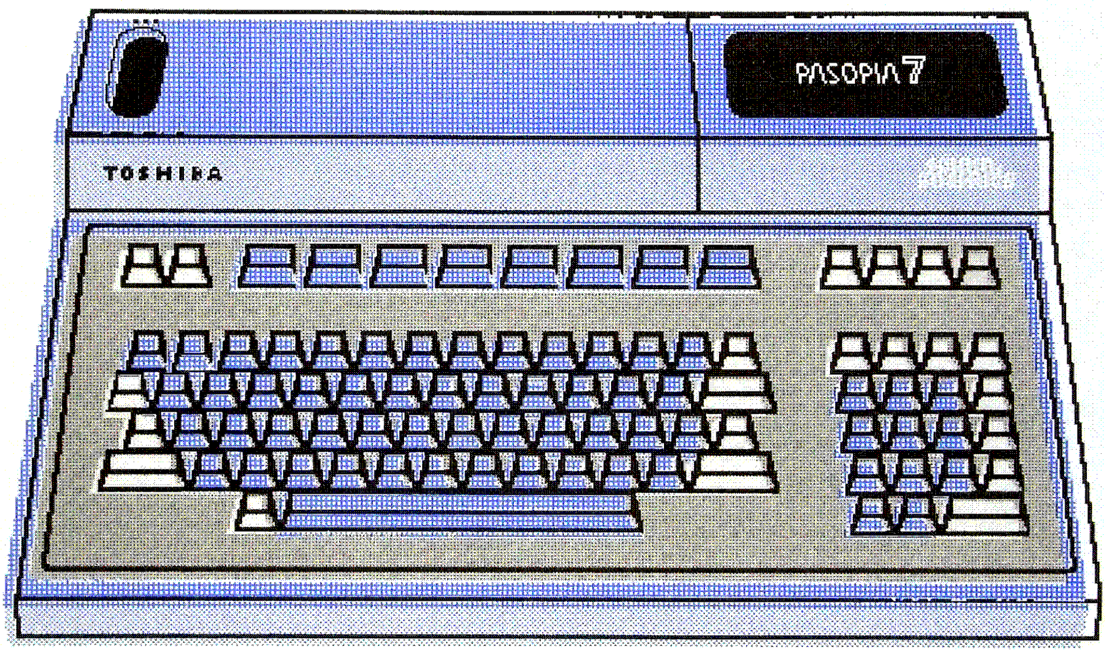


Illustration
TOSHIHITO ITO

1982年，東芝は“パソピア”（163,000円）をもってパソコン業界に参入を果たした．この機械は当時としてはめずらしく，当初からパソコンとして企画されたものであって，ICの拡販が目的ではなかったように見える．というのも，これはオフィスオートメーション事業部の作品であり，使っているICを見ても，TTLの多くはTI（テキサス・インスツルメンツ）社製であり，自社のものの利用度は低い．CPUチップもシャープのZ-80Aであり，最も人気のあるICを選定している．

実際，社内にはすでに多くのICニーズがあった．たとえばワープロでは日本初の製品であるJW-10（1987年発表，価格は625万円）以来のシリーズが続いているし，ビジネスパソコンシリーズもBP-100（1980年発表，価格は140万円）以来続いていた．そのうえ，オフコン，ミニコンなどのTOSBACシリーズもあった．このような社内環境の中で発表されたのが“パソピア”なのである．

そのため，特徴としては“プロが作った本格派パソコン”ということになり，ゲーム，ホームユース以外に科学技術者向けを狙っていた．同時期に家庭機器事業部

からも“パソピアミニ”(54,800円)が出されているが，こちらのほうは，RAM が4KB しかなく，表示も1行しかないため，キーボードのみフルキーボードが付いていたが，どちらかというと電卓に近い印象があった．しかし，後のパソコンの小型化技術開発への大きな布石としての役目を果たすことになる．

翌1983年に改良版として“パソピア7”(119,800円)を出した．パソピア7になると，漢字・ひらがなのディスプレイ用にはファミコン式のROMパックを使っていた．おもしろいことに，当時からすでに500g と軽量の“液晶ディスプレイ”(40,000円)が用意されていた．そのほかに，本体，とくにキーボードまわりの色がアースブラウン，ワインレッド，ブルーの3色が用意されていて，簡単に取り外しできたのも特徴であろう．

使用しているICの部品にしても，ROMは2764×7個，RAMはD4164C×8個という具合に，当時の最新技術が織り込まれていたが，この点については，とくにPRするようなこともなく，グラフィックス，構造化プログラミング，音楽演奏などがPRの重点になっていたように思えた．また，パソピア7になると，本体に付いているキーボードの上部に“SOUND GRAPHICS”と，利用分野について念を押している．このような例はめずらしい．

ソフトウェアでは，立ち上げ時の読み込み用にT-BASICタイプとOA-BASICタイプのROMの2種類があり，ディスク版もあった．ROMパックでMINI-PASCALも供給された．フロッピーディスクは5インチ両面倍密の280KB(168,000円)，および片面倍密の140KB(79,000円)タイプで，CP/Mを使うように考えられていた．おもしろいのは，パソピア7用に開発された“T-BASIC7”で，“ディスクBASIC”とはいわずに，すなわちROM BASICによりディスクの制御が行えたことである．

さて，東芝という会社の社風は，その生い立ちからも類推できよう．創業は明治8年“からくり屋儀右衛門”の名で知られる田中久重氏によるものといわれている．すでに100年以上たっているとはいえ，その伝統は続いていて，“からくり”を“新技術”と読みかえると，そのようなものを一生懸命追究している人が多くいるのではないか．

コンピュータ用のICにしても，インテル社が8080を発表したのと同じ1974年に，すでにTLCS-12というシステム用に“T3153”という12ビットのCPU，ならびにその周辺チップを発表している．12ビットというと，現在ではめずらしいものであるが，当時は未だCPUチップが何に使われるようになるか議論の絶えないところであった．主要市場は電卓，時計，自動車などと考えられており，4ビットは単純なデータ処理に，8ビットは英数字を扱う端末とか周辺機器用に，12ないし16ビットは自動制御用や科学技術計算用などに使われるだろうとかいわれていた．実際，だいたいはこのようになってきたようである．

その後の発展であるが，1985年SORD社が傘下に入り，1986年には海外で販売していたIBM PC/AT互換機であるT-3100に日本語処理機能をもたせたJ-3100(498,000円)を出し，“ラップトップブーム”を作り出した．

1989年に入ると，J-3100SS“DynaBook”(198,000円)をヒットさせた．これはA4版程度のサイズで，厚さ44mm，2.7Kg というノートブック型で，一太郎などのワープロソフトも走るということで，新しいパソコンのジャンルを作り出した．大ヒットの前には長い歴史を必要としているのかもしれない．

●パソピア拡張ユニット外部バス

番号	信号名	I/O	番号	信号名	I/O
1	GND		2	GND	
3	＊SC8MHz	O	4	GND	
5	GND		6	＊MI	O
7	GND		8	＊IORQ	O
9	GND		10	＊RESET	O
11	GND		12	＊RD	O
13	GND		14	＊WR	O
15	GND		16	＊INR	I
17	GND		18	＊NMI	I
19	GND		20	DB0	I/O
21	GND		22	DB1	I/O
23	GND		24	DB2	I/O
25	GND		26	DB3	I/O
27	GND		28	DB4	I/O
29	GND		30	DB5	I/O
31	GND		32	DB6	I/O
33	GND		34	DB7	I/O
35	GND		36	AD0	O
37	GND		38	AD1	O
39	GND		40	AD2	O
41	GND		42	AD3	O
43	GND		44	AD4	O
45	GND		46	AD5	O
47	GND		48	AD6	O
49	GND		50	AD7	O

＊：負信号

祐 安 重 夫

プログラマの道具学

第14回　時計

　中学生になった時に，入学祝に腕時計をもらったが，高校生になった頃には腕時計をする習慣をなくしていた．汗かきで皮膚があまりつよくなかったため，腕時計をしているとすぐにかぶれてしまうためだった．

　街中には銀行の建物や電車の駅など，いたるところに時計があったし，バスにのれば運転席のそばに時計がついていたので，時計をもってあるかなくても別にこまることはなかった．一時期，あるいている時にそばをとおりかかったバスをのぞきこんで，その中の時計の時間をみることを自分の特技だといって，顰蹙をかっていたこともある．

　懐中時計をもちあるいていた時期もあったが，ポケットにいれておくと結構かさばるし，何度か紐をきっておとしてしまい，修理にだしたりしているうちに，これもいつの間にかつかわなくなってしまった．

　時計が街中にあふれているという状態は，その後もかわらなかったが，マイクロプロセッサが普及するとともに，街中の時計が少しづつディジタル化していった．人間の思考の基本にあるのはあくまでも直観であるというのが，かねてよりの持論であったこともあり，ディジタル時計というのはなんとなく不愉快な代物だった．そのかんがえはいまでもかわっていない．

　それでも仕事の都合で，時計をもっていないと不便な時期があり，時計つきの電卓をポケットにもちあるいていたこともあった．もちろん，大きらいなディジタル時計である．これも仕事の環境がかわったこともあって，3回目の電池交換を前にして，すてられていった．海外旅行に出かける時には，世界時計の機能のある電卓をもっていったが，日本では机の中にしまったままである．

　現在の仕事場の壁には普通のアナログ式の時計がかかっているが，コンピュータにむか



Illustration
Mikio Sasaki



って仕事をしている時は，ビットマップ・ディスプレイ上に表示された時計をみてしまうことがおおい．X-Windowの画面の右下にディジタル時計を，右上にアナログ時計を表示しているが，時間はもちろんアナログ時計でみる．ディジタル時計をだしてあるのは，今日の日付を確認する時のためである．

それ以外にもディジタル時計なら，いたるところにころがっている．時計として存在しているのではなく，マイクロプロセッサが内蔵された電気製品に，たいていはいっているのである．ファックス，VTR，電話から，エアコン，洗濯機，電子レンジ，炊飯器といったものまで，家中いたるところに時計があるのだが，それらは時計という気がまったくしない．現在つかっている電話にいたっては，ボタンをおすと合成音で日付と時間をおしえてくれさえする．

各種の電気製品が時計を内蔵しているのは，時間を指定して自動的にコントロールするためだが，実際に生活しているとVTR以外にはその機能をつかうことはあまりない．多分それは時計をもちあるかない生活が可能なことと，どこかでつながっているのだろう．その意味では，あまり現代的な生活をしているとはいえない．

一歩外にでると，停留所でバスをまっていても，時刻表どおりにバスがくることはほとんどない．東京の交通事情はバスを時間どおりにはしらせることなど，とっくの昔に不可能にしてしまっている．正確な時間を数字でしめしてくれるものが，いたるところにありながら，正確な時間どおりに物事が進行していかないというのが，世の中で技術の進歩とよんでいるものの正体なのかもしれない．

プログラム遊工房

谷口典生

拾参乃巻
「colax／落下型パズル」ゲーム



今回は，落下型のパズルゲーム（テトリスのようなタイプのゲーム）を作ってみました．コラムスとクラックスのあいのこのようなゲームになったので，colax（コラックス）と名付けました．ワンポイントテクニックでは，プログラムの改良方法について具体的な例をあげて説明しています．

リスト1 colax. C

```
/*
 * colax.c                  玉落としゲーム
 */

#include <stdio.h>
#include <time.h>
#include <dos.h>

#ifndef __TURBOC__          /* Turbo C → MS-C(Quick C) 関数名変換 */
#define enable()            _enable()
#define inportb(x)          inp(x)
#define outportb(x,y)       outp(x,y)
#define setvect(x,y)        _dos_setvect(x,y)
#define getvect(x)          _dos_getvect(x)
#endif

#define XSIZE               7                       /* 横サイズ */
#define YSIZE               10                      /* 縦サイズ */
#define XBASE               5                       /* 横表示位置 */
#define YBASE               5                       /* 縦表示位置 */
#define XLEN                70                      /* ベルトコンベアの長さ */
#define MAXCOL              5                       /* 色数 */
#define MAXBALLS            15                      /* 同時に動く玉の最大数 */
#define gotoxy(x,y)         printf("¥x1b[%d;%dH",y,x)

unsigned int (far *ptext)[80];  /* VRAMテキスト文字コードエリアへの配列 */
unsigned int (far *pattr)[80];  /* VRAMアトリビュートエリアへの配列 */

int btable[XSIZE][YSIZE];                           /* 積まれた玉 */
int ballx[MAXBALLS],ballcol[MAXBALLS];              /* 動いている玉のX座標と色 */
int xopen;                                          /* 蓋が開いている場所 */
enum color {BLACK, BLUE, RED, MAGENTA, GREEN, CYAN, YELLOW, WHITE};
int attrcode[8] = {0x01, 0x21, 0x41, 0x61, 0x81, 0xa1, 0xc1, 0xe1};
int score;                                          /* 得点 */
long counter=0,lastcount;                           /* VSYNC割り込みカウンタ */
void (interrupt far *oldvsync)();                   /* 元のVSYNC割り込みルーチン */

void interrupt far vsync()                          /* VSYNC割り込みルーチン */
{
        counter++;                                  /* カウンタ値を増やす */
        oldvsync();                                 /* 元のVSYNC割り込みを呼び出す */
        outportb(0x64, 0);                          /* VSYNC割り込みをリセット */
}

void set_vsync()                    /* 割り込みルーチンの設定 */
{
        outportb(2, inportb(2) | 4);                /* VSYNC割り込みをマスク */
        oldvsync = getvect(0x0a);                   /* 元の割り込みベクタを保存 */
        setvect(0x0a, vsync);                       /* vsync() をベクタに設定 */
        outportb(2, inportb(2) & ~4);               /* マスクのクリア */
        outportb(0x64, 0);                          /* VSYNC割り込みのリセット */
        enable();                                   /* 割り込み許可フラグをセット */
}

void restore_vsync()                /* 元の割り込みルーチンの復帰 */
{
        outportb(2, inportb(2) | 4);                /* VSYNC割り込みをマスク */
        setvect(0x0a, oldvsync);                    /* 割り込みベクタを元に戻す */
}

void putxy(int x, int y, int col, int code)         /* 文字の直接書き込み */
{
        ptext[y][x] = code;
        pattr[y][x] = attrcode[col];
        if(code != (code & 0xff)) {                 /* 2バイトコードか? */
                ptext[y][x+1] = code + 0x80;
                pattr[y][x+1] = attrcode[col];
        }
}

void init()                         /* 変数と画面の初期化 */
{
        int x,y,i;

        for(x=0; x<XSIZE; x++)
                for(y=0; y<YSIZE; y++)
                        btable[x][y] = 0;
        for(i=0; i<MAXBALLS; i++)
                ballx[i] = -1;

        ptext = (void far *)0xa0000000L;            /* テキストエリア */
        pattr = (void far *)0xa2000000L;            /* アトリビュートエリア */

        printf("¥x1b*¥x1b[>5h");                    /* 画面クリア&カーソル消去 */
        gotoxy(XBASE+4, YBASE+2);
        for(x=0; x<XLEN-2; x++)
                cputs("￣");
        for(y=0; y<=YSIZE; y++)
                for(x=0; x<=XSIZE; x++)
                        putxy(XBASE+x*3, YBASE+y+1, WHITE, '｜');
```

プログラムの使い方

このプログラムはPC-9801シリーズ用で，Turbo C, QuickCおよびMS-Cに対応しています．リスト1を打ち込み，コンパイルして実行ファイルを作ってください．

プログラムの実行はそのまま

colax

としてください．引数は必要ありません．

ゲームの遊び方

ゲームを開始すると，右側から色のついた玉が流れてきます．画面左側には7つの縦穴があり，いずれかの縦穴(最初は左から2番目)の上に逆三角形のマーカーがあります．このマーカーは［4］または［H］のキーで左に，［6］または［L］で右に動かすことができます(端を越えると逆の端に移ります)．マーカーが指している縦穴は上が開いていて，流れてきた玉が落ちるようになっています．ただし，左端

```
 91:         for(x=0; x<XSIZE; x++) {
 92:                 putxy(XBASE+x*3, YBASE+y+1, WHITE, '+');
 93:                 putxy(XBASE+x*3+1, YBASE+y+1, WHITE, '-');
 94:                 putxy(XBASE+x*3+2, YBASE+y+1, WHITE, '-');
 95:         }
 96:         putxy(XBASE+x*3, YBASE+y+1, WHITE, '+');
 97:         xopen = 1;
 98:         putxy(XBASE+xopen*3+1, YBASE-1, WHITE, 0x2702);         /* ▼ */
 99:         putxy(XBASE+xopen*3+1, YBASE+1, WHITE, 0x2101);         /* 空白 */
100:         score = 0;
101:         gotoxy(XBASE+40, YBASE+9);
102:         printf("SCORE :%5d", score);
103:
104:         set_vsync();                                    /* VSYNC割り込みをセット */
105:         lastcount = counter;                            /* 最終カウンター値 */
106: }
107:
108: void deinit()                   /* プログラム終了時の処理 */
109: {
110:         restore_vsync();                                /* VSYNC割り込みを復帰 */
111:         printf("¥x1b[>5l");                             /* カーソル表示 */
112:         gotoxy(1, 23);
113: }
114:
115: void keyin()                    /* キー入力 */
116: {
117:         int xnew;
118:
119:         xnew = xopen;
120:         if(kbhit()) {
121:                 switch(getch()) {
122:                 case '4':  case 'h':  case 'H':         /* 左へ移動 */
123:                         xnew = xopen==0 ? XSIZE-1 : xopen-1;
124:                         break;
125:                 case '6':  case 'l':  case 'L':         /* 右へ移動 */
126:                         xnew = xopen==XSIZE-1 ? 0 : xopen+1;
127:                         break;
128:                 case 0x1b:  case 'q':  case 'Q':        /* ゲーム中断 */
129:                         deinit();
130:                         exit(0);
131:                 case 'p':  case 'P':                    /* ゲーム一時停止 */
132:                         getch();
133:                         return;
134:                 default:
135:                         return;
136:                 }
137:                 putxy(XBASE+xopen*3+1, YBASE-1, WHITE, 0x2101); /* 空白 */
138:                 if(xopen!=0) {
139:                         putxy(XBASE+xopen*3+1, YBASE+1, WHITE, '^');
140:                         putxy(XBASE+xopen*3+2, YBASE+1, WHITE, '^');
141:                 }
142:                 xopen = xnew;
143:                 putxy(XBASE+xopen*3+1, YBASE-1, WHITE, 0x2702); /* ▼ */
144:                 putxy(XBASE+xopen*3+1, YBASE+1, WHITE, 0x2101); /* 空白 */
145:         }
146: }
147:
148: void wait(int val)                      /* 時間待ち */
149: {
150:         while(counter-lastcount<val)
151:                 keyin();
152:         lastcount = counter;
153: }
154:
155: void generateball()                     /* 新しい玉を生成 */
156: {
157:         int i;
158:
159:         for(i=0; i<MAXBALLS; i++)
160:                 if(ballx[i]==-1)                /* 配列の空きを捜す */
161:                         break;
162:         if(i!=MAXBALLS) {
163:                 ballx[i] = XLEN;                        /* X座標 */
164:                 ballcol[i] = (rand() % MAXCOL) + 1;     /* 色 */
165:         }
166: }
167:
168: int issame(int x, int y, int col)               /* 色コードが同じか? */
169: {
170:         if(col==0 || x<0 || x>=XSIZE || y<0 || y>=YSIZE)
171:                 return(0);                      /* 配列の外なら不一致 */
172:         return((col&7)==(btable[x][y]&7));      /* 下位3ビットのみ比較 */
173: }
174:
175: int check(int mag)                      /* 同色の玉の連続状態を調べる */
176: {
177:         int x,y,c,tmpy;
178:         int result=0;
179:
180:         for(x=0; x<XSIZE; x++)
181:                 for(y=0; y<YSIZE; y++) {
182:                         c = btable[x][y];
183:                         if(issame(x+1,y,c) && issame(x+2,y,c)) { /* 横 */
184:                                 btable[x][y] |= 8;
185:                                 btable[x+1][y] |= 8;
186:                                 btable[x+2][y] |= 8;
```

の縦穴はつねに開いています.

マーカーを動かして任意の穴に玉を落としていきますが，同じ色の玉が縦横斜めに3つ以上並ぶとその玉は消え，得点が加算されます（1つにつき1点）．消えた場所より上にあった玉は下に落ちてきますが，この動作によって再度同じ色の玉が並ぶと，2度目に消えた玉については点数が2倍になります（さらに連鎖反応が起これば，3倍4倍となります）．

獲得した点数は画面右下に表示されますが，一定の点数に達すると流れてくる玉の速度が上がります．速度が変わる点数は50点，100点，300点，800点……，となっています（リスト1の286行目で定義されている）．

それぞれの縦穴には10個まで玉が入りますが，どれかの穴に11個目の玉が落ちるとゲームオーバーになります．ゲームを途中でやめたいときは[ESC]か[Q]のキーを押してください．また，[P]のキーでゲームを一時停止することができます（任意のキーで再開します）．

プログラムの説明

vsync()

VSYNC割り込みルーチン．VSYNC割り込みが発生するごとにカウンタ値を1増やします．

```
187:                         }
188:                         if(issame(x+1,y+1,c) && issame(x+2,y+2,c)) { /* 斜め */
189:                                 btable[x][y] |= 8;
190:                                 btable[x+1][y+1] |= 8;
191:                                 btable[x+2][y+2] |= 8;
192:                         }
193:                         if(issame(x-1,y+1,c) && issame(x-2,y+2,c)) { /* 斜め */
194:                                 btable[x][y] |= 8;
195:                                 btable[x-1][y+1] |= 8;
196:                                 btable[x-2][y+2] |= 8;
197:                         }
198:                         if(issame(x,y+1,c) && issame(x,y+2,c)) { /* 縦 */
199:                                 btable[x][y] |= 8;
200:                                 btable[x][y+1] |= 8;
201:                                 btable[x][y+2] |= 8;
202:                         }
203:                 }
204:         for(x=0; x<XSIZE; x++)          /* 消す玉の数をカウント */
205:                 for(y=0; y<YSIZE; y++)
206:                         if(btable[x][y]>=8) {
207:                                 result++;
208:                                 putxy(XBASE+x*3+1, YBASE+y+2, WHITE, 0x2802);
209:                         }
210:         if(!result)
211:                 return(result);         /* 消す玉がなければ戻る */
212:
213:         wait(20);                       /* 時間待ち */
214:
215:         for(x=0; x<XSIZE; x++) {        /* 玉を下につめる */
216:                 tmpy = YSIZE-1;
217:                 for(y=YSIZE-1; y>=0; y--) {
218:                         if(btable[x][y]<8)
219:                                 btable[x][tmpy--] = btable[x][y];
220:                         else
221:                                 putxy(XBASE+x*3+1, YBASE+y+2, WHITE, 0x2101);
222:                 }
223:                 for(y=tmpy; y>=0; y--)
224:                         btable[x][y] = 0;
225:         }
226:
227:         wait(16);                       /* 時間待ち */
228:
229:         for(x=0; x<XSIZE; x++)
230:                 for(y=0; y<YSIZE; y++)
231:                         putxy(XBASE+x*3+1, YBASE+y+2, btable[x][y], 0x7c01);
232:         score += result * mag;
233:         gotoxy(XBASE+47, YBASE+9);
234:         printf("%5d", score);           /* スコア表示 */
235:         return(result);
236: }
237:
238: void fallball(int n)            /* 玉を落とす */
239: {
240:         int x,y,flag=1;
241:         int mag=1;              /* 得点の倍率(連鎖消しで上がる) */
242:
243:         x = ballx[n] / 3;
244:         y = 0;
245:         while(flag) {
246:                 putxy(XBASE+ballx[n]+1, YBASE+y, ballcol[n], 0x7c01);
247:                 wait(1);
248:                 if(y>0 && btable[x][y-1]!=0 || y>YSIZE) {
249:                         flag = 0;
250:                 } else {
251:                         putxy(XBASE+ballx[n]+1, YBASE+y, WHITE, ' ');
252:                         putxy(XBASE+ballx[n]+2, YBASE+y, WHITE, ' ');
253:                         y++;
254:                 }
255:         }
256:         if(y==1) {                      /* 上まで積み上がった */
257:                 deinit();
258:                 exit(0);                /* ゲームオーバー */
259:         }
260:         btable[x][y-2] = ballcol[n];
261:         ballx[n] = -1;
262:         while(check(mag++));    /* 倍率を加算しながらcheck()を呼び出す */
263: }
264:
265: void moveball()                 /* 玉を動かす */
266: {
267:         int n;
268:
269:         for(n=0; n<MAXBALLS; n++) {
270:                 if(ballx[n]==-1)                /* 使われていない */
271:                         continue;
272:                 putxy(XBASE+ballx[n], YBASE, WHITE, ' ');
273:                 putxy(XBASE+ballx[n]+1, YBASE, WHITE, ' ');
274:                 ballx[n]--;                     /* 左へ動かす */
275:                 if(ballx[n]==0 || ballx[n]==(xopen*3))
276:                         fallball(n);            /* 穴の上にきたら下へ落とす */
277:                 else
278:                         putxy(XBASE+ballx[n], YBASE, ballcol[n], 0x7c01);
279:         }
280: }
281:
282: void main()                     /* メインルーチン */
```

set_vsync()
restore_vsync()

VSYNC割り込みルーチンの設定，復帰をします．

putxy()

テキスト画面に直接文字を書き込みます．

init()

変数の初期化と初期画面の作成を行います．

81-82行

farポインタptext，pattrにテキストVRAMのアドレスを設定します．

84-102行

画面クリアとカーソル消去を行い，ゲーム画面を描きます．

deinit()

プログラムを終了する前に呼ばれるルーチンで，VSYNC割り込みの復帰とカーソルの表示を行います．

keyin()

キー入力を調べ，マーカーの移動やゲームの中断，一時停止の処理をします．

wait()

VSYNC割り込みのカウンタを使い，ゲームの進行速度を一定にするための時間待ちをします．時間待ちの間，つねにキー入力ルーチンを呼び出すことによって操作性を向上させています．

generateball()

新しく玉を生成します．玉のX座標の配列ballx[]から空きを捜し，X座標の初期値と色コードを設定します．

```
283: {
284:         int term=10;                    /* 次に玉が出るまでの時間 */
285:         int waitval=6;                  /* 時間待ちの長さ(=ゲームスピード) */
286:         int wtable[7] = {10000, 5000, 2000, 800, 300, 100, 50};
287:         time_t seedtime;
288:
289:         time(&seedtime);
290:         srand((unsigned int)seedtime);          /* ランダマイズ */
291:         init();                                 /* 変数と画面の初期化 */
292:         while(1) {
293:                 keyin();                        /* キー入力 */
294:                 if(--term<=0) {
295:                         generateball();                 /* 新しく玉を生成 */
296:                         term = (rand() % 20) + 5;       /* 次の玉が出る間隔 */
297:                 }
298:                 moveball();                     /* 玉を動かす */
299:                 if(waitval>=0 && score>wtable[waitval])
300:                         waitval--;              /* 得点によりスピードアップ */
301:                 wait(waitval);                  /* 時間待ち */
302:         }
303: }
```

issame()

引数colとbtable［x］［y］の下3ビットを比較し，色コードが一致するかどうかを返します．指定したXY値が配列の範囲内にあるかどうかもチェックします．

check()

同じ色の玉の連続状態を調べ，3つ以上並んでいるものを消去し，画面表示と得点の加算を行います．

180-203行

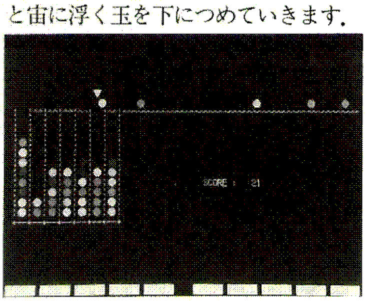
縦，横および斜め2方向について，3つ以上同じ色の玉が並んでいる場合は8とORをとってビット3を立てておきます．

204-211行

再度配列を順番に調べ，消去するべき玉(＝ビット3が立っているもの)の数をカウントし，該当する玉を米印にして表示します．消去する玉がないときは0を返して戻ります．

215-225行

配列btableの中で，玉を消去したあと宙に浮く玉を下につめていきます．



同時に米印で表示した玉に空白を上書きしていきます．

229-235行

玉をすべて描き直し，スコアを加算して表示し，消した玉の数を返します．

fallball()

穴の上にきた玉を下に落とす処理を行います．

243-255行

すでにある玉にぶつかるか縦穴の底に達するまで玉を下へ移動させます．

256-262行

上まで積み上がったらゲームオーバーとし，そうでなければ配列に玉を加えて，消える玉がなくなるまでcheck()を呼び出します．

moveball()

玉を右から左へと移動させます．開いている穴の上にきた玉はfallball()を呼び出して落下させます．

main()

294-297行

不定な間隔で新しく玉を生成します．

299-300行

得点が一定の値に達するごとに時間待ち変数を減らしてゲーム速度を上げます．

ワンポイントテクニック

プログラムを改造する

この連載のねらいはプログラミングのおもしろさを発見してもらうところにあります．リストをそのまま打ち込んで(付録ディスクがあるので手で入力する必要さえないのですが)コンパイルして遊んでみるだけで終らせずに，プログラムの動きを理解してもらいたいと思います．そのためにはただリストを眺めるだけでなく，部分的にプログラムを変更してみて，実際の動きがどう変わるか試してみるのが効果的です．

もしC言語の文法をまるで知らなくても，数値を変更するくらいなら簡単です．たとえば今回のプログラムでは，17行目から各種定数を定義していますが，#define文が定数を定義する文であることを知らなくても，コメントに「/＊ 横サイズ ＊/」と書いてあることから，その前の「7」という数字が何かの横方向の長さを示しているらしいと予想できます．そう思ったら，とりあえず「7」を「8」や「6」に変えてコンパイルし，実行してみます．縦穴の数が変わっていることに気付いたら，もっと違う数にしてみたり，ほかの定数を変更してこんどはどこが変わるかを観察したりします．

プログラムによっては，1つの定数を変えただけではうまく動かなかったりするかもしれません．大きすぎる数

や小さすぎる数にしたら暴走してしまうこともあるかもしれません．しかし，うまくいかないかもしれないことを恐れずに，「とりあえずやってみる」ことが大切なのです．

少しC言語がわかってきたら，もっと大きな改造を試みるようにします．ここに掲載するプログラムはあまり行数が増えないように書いているため，どうしても不完全なものになってしまいます．今回のゲームでは，レベルの選択やゲームオーバー後の処理（ハイスコアの登録など）などがありません．画面も少しさびしいような気がします．そういった部分をうまく仕上げると，ゲームとしての見栄えもだいぶよくなると思います．

以下に，具体的な改造方法の例をあげてみることにします．

例その1「魔宝石を出す」

セガ版のコラムスに出てくる魔宝石のようなものを出してみます．

まず，通常の玉に混ざって魔宝石となる玉が出現するようにします．ここでは単純に白い玉を魔宝石とすることにしましょう．リスト1のgenerateball()内，164行目の下にリスト2のようにつけ加えます．これで100分の1の確率で白い玉が生成されることになります．ランダムではなく，生成された玉をカウントして，一定数ごとに魔宝石が出るようにしてもおもしろいでしょう．

次に，白い玉が落ちてきたときは，下にある玉と同じ色の玉をすべて消してしまうようにします．このためにはリスト1のfallball()の最初で変数の宣言を追加しておき，261行目の下にリスト3をつけ加えます．

ゲームは少しやさしくなるので，22行目で定義している色数を6にして少しゲームの難易度を上げたほうがバランスがとれるかもしれません．白の玉を魔宝石として使っている関係上，色数を7にしてしまうと不都合が出るので，色数を7に増やしたいときは，魔宝石は異なるキャラクタで表示する必要があります．

例その2「白黒ディスプレイ対応」

今回のゲームでは玉を色で区別していますが，98noteなどで白黒ディスプレイを使っている場合は玉の色が見分けにくいと思います．そこで，色でなく形を変えて玉を表示するようにプログラムを改造してみましょう．

まずプログラムの頭のほうで，リスト4のように各色に対応した文字コードを定義しておきます．

玉はputxy()を使って表示しているので，putxy()の関数の最初(63行目の前)でリスト5のように文字コードを変換してしまいます．ここで使っている文字コードはテキストVRAMに書き込むコードで，JISコードの上位バイトと下位バイトを交換してから0x20を引いたものです(たとえば黒丸のJISコード0x217c→0x7c21→0x7c01)．

リスト2

```
if((rand()%100)==0)
        ballcol[i] = WHITE;
```

リスト3

```
if(ballcol[n]==WHITE) {
        btable[x][y-2] |= 8;
        if(y<=YSIZE) {
                c = btable[x][y-1];
                for(x=0; x<XSIZE; x++)
                        for(y=0; y<YSIZE; y++)
                                if(btable[x][y]==c)
                                        btable[x][y] |= 8;
        }
}
```

リスト4

```
int bcode[8]={0x2101, 0x7b01, 0x7c01, 0x7e01, 0x2102, 0x2202, 0x2302, 0x7a01};
```

リスト5

```
if(code==0x7c01) {
        code = bcode[col];
        col = WHITE;
}
```

IC千夜一夜

第8回 ビット・スライス型マイクロプロセッサとマイクロプログラム ―その1―



虎岩 登

今回から2回にわたって，ある特殊な種類のマイクロプロセッサ，ビット・スライス型のマイクロプロセッサについてお話することにします．先に断わっておきますが，ビット・スライス型というのは，CPUを組み立てるための部品の構成法を表す用語です．つまり，これからの2回は特定のCPUについて述べるのではなく，CPUを組み上げるための部品について語ることになります．

現在では，処理能力の非常に大きいコンピュータまで1チップのCPUになっていますが，いまから約20年もさかのぼればそもそもマイクロプロセッサ自体がありませんでしたし，その後8bitのマイクロプロセッサが一般的になってきた時代にも，ミニコンピュータ以上では完全な1チップ構成のCPUが使えませんでした．このテーマはその頃のミニコンピュータなどに使われていた技術だと思ってください．残念ながら私にはそれ以上の規模のコンピュータに関する知識がありませんので．

では，なぜ1チップにできなかったのでしょうか．当時のマイクロプロセッサは(いまでも変わらないのですが)機器制御など組み込み用途を目的としていて，処理速度が遅くても実用になる分野によく使われていました．しかし，ミニコンピュータは汎用のコンピュータとして使われることも多く，マイクロプロセッサ以上の性能や拡張性を当然のように要求されます．そうなると高速のバイポーラ・プロセス，つまりTTLやECLといったプロセスを利用することになります．これらはたしかにマイクロプロセッサに広く使われていたMOSプロセスに比べて高速なのですが，消費電力も大きく，さらにMOSプロセスよりずっと集積度が低いという難点がありました．結局，当時のIC製作技術からすれば，ミニコンピュータで一般的な16bitのCPU，ないし32bitのCPUは，複雑すぎて1チップの中に集積できなかったのです．

このように，比較的小規模なICを組み合わせてCPUを設計していた時代には，CPU構成用の専用ICが誕生して当然です．その代表的な答えが，ビット・スライス型マイクロプロセッサです．また，ビット・スライス型マイクロプロセッサを利用してCPUを構成する場合，一般的にはマイクロプログラム方式と呼ばれる制御方式が用いられていました．このマイクロプログラムという言葉は，最近でもCISC/RISC論争などで耳にする機会もあり

ますし，意外とCPUだけでなく一般の周辺LSIの内部にも使われていることがありますので，これについては次回に触れてみようと思います．

CPUの構造と設計の考え方

ビット・スライス型誕生以前

ビット・スライス型マイクロプロセッサは，それ以前の個別のICを使ってCPUを構成していた時代の一般的な考え方がある種の進化を遂げて誕生したものです．そこで，まずはビット・スライス誕生前からの設計の考え方を見ていきましょう．

そもそもCPUとはどういう構造になっているのでしょうか．大まかな構造は図1のようになっています．機械命令を解釈し，実行するハードウェアがCPUなのですが，そのためには命令を解釈する部分が当然必要になります．さらに演算処理を行う部分(ALU)とレジスタ類が含まれ，また外部（バス）とのインターフェース部分と全体のタイミングを制御する部分も必要です．一部のCPUでは内部にレジスタを含まないアーキテクチャもありますが(演算命令などに2つないし3つのアドレスを指定して必ずメモリ-メモリ間で演算を行うものとか，メモリの一部をレジスタのように扱うもの)，ここではレジスタを含むモデルで考えましょう．

これら全体を設計する場合，全体をひとつのチップに組み込むことができないとすれば，幾つかのブロックに分けて設計しようと考えるのが普通でしょう．

最初に，最もコンピュータらしい演算部分について，具体的に考えていきます．この部分は難しい言葉では算術論理演算装置と呼ばれています．英語ではArithmetic and Logic Unitですから，省略形としてALUと表記することにします．ALUという名の通り，加減算のような算術演算や，AND OR NOTのような論理演算を行います．ALUで処理される演算の大部分は2項演算ですから，入力が2系統に出力が1系統，さらにどのような演算を行うかの司令が1系統必要なユニットです．ここで16bitのCPUを設計する場合を考えると，データの入出力だけで16×3，48本も信号が必要になります．さらに司令系統やその他のコントロール線が10本程度必要になると60本近い端子が必要なユニットになります．また回路も十分すぎるほど複雑になってしまいますから，ALUだけでも1チップにするのは困難です．したがって，さらにALUをいくつかに分割する必要が出てきます．

図1　CPUの内部構造

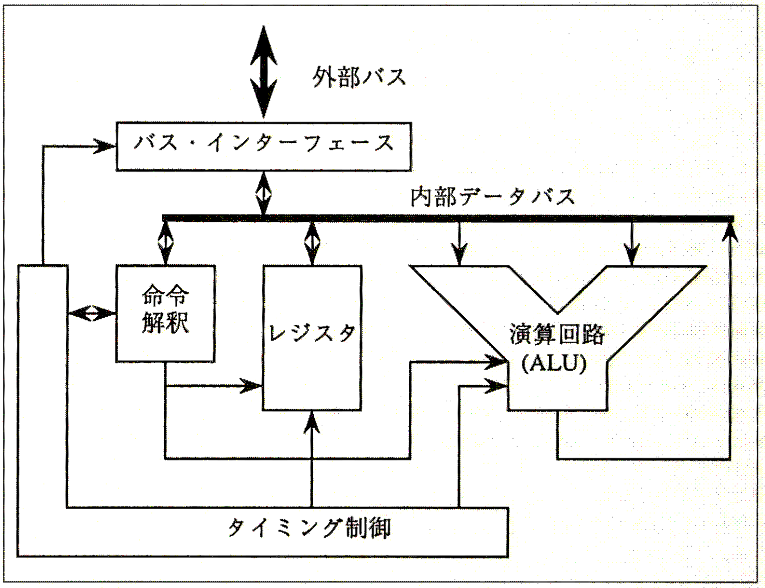
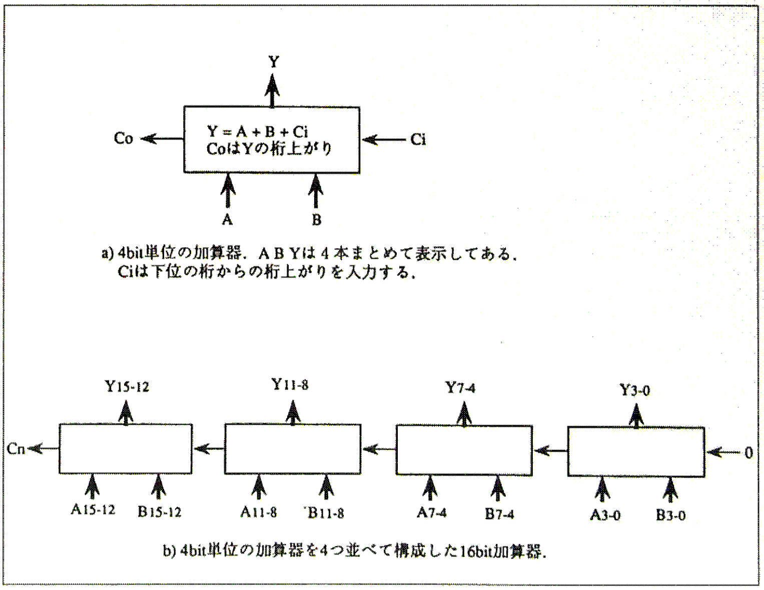


図2　4bit同士の加算回路



唐突ですが，もし8086の16bitの演算命令を使わずに16bitの加算を行うにはどうしたらよいでしょうか．数値はAXとBXレジスタに入っていることとしましょう．そうすると，

```
ADD   AL, BL
ADC   AH, BH
```

というように2回に分けて加算を行うことになりますね．ただし2回目は下の桁からの桁上がりも一緒に足しこみます．同じように8bitの演算命令だけで，32bitや64bit長のデータを加算することもできます．自分より下位の桁から発生する桁上がりを同時に加えてやればよいのです．これは，ハードウェアで加算を行う場合でも同じことがいえます．図2のように，4bitの数同士の加算回路があれば，自分より下位の桁からの桁上がりを同時に入力してやるだけで，16bitでも64bitでも必要なビット長のデータを加算する回路を構成できます．

減算も同じ要領で実現できます．また，ALUで実行すべき論理演算は，各ビットごとの演算ですから，他の桁とは無関係に実行できます．

ですから，ALUを1チップに入れることが無理なら，4bit程度の単位に区切ってひとつのICにすれば，それを用いて容易に任意のビット長のALUが実現できることがわかります．

では，具体的なALU専用ICの例を見てみましょう．テキサス・インスツルメンツ社のTTL-ICファミリの中の，SN74181がその代表的なものでしょう．写真1は，昔のミニコンピュータのCPUに使われていた基板の一部ですが，しっかりと4個並んで使われています．入力は4bitのデータ入力が2系統，出力も4bit，さらにどのような演算を行うのかの司令線が5本，残りは電源と桁上がり関係です．このICで行うことのできる演算を表1に示します．CPUに必要な演算をひと通り備えていることがわかります．一部，何に使うのか想像もつかない演算がありますが，これは別の演算を行うための副作用みたいなもので，実際には使わなければよいのです．

さて，ALUは少ないビット幅のものを並べれば実現可能であることがわかりました．では他の部分はどうでしょうか．レジスタ部は，いってみれば小容量のRAMです．ですから，たとえば4bit単位のメモリを単純に4個並べれば16bit幅のメモリを構成できるように，簡単に多ビットのレジスタに拡張できます．あと，図1にははっきりと書いてはいませんが，ALUの入出力を切り替える部分も同様に拡張できます．図では単にバスラインのように書いてありますが，レジスタからのデータや外部バスを通して読み込んだメモリのデータなどを適宜ALUに接続し，また演算結果を書き込むためのデータの通り道を選択するための，切り替えスイッチ部分です．基本的に他のビットとは関係がありませんから単に並べるだけで済みます．もっとも，設計に

写真1　あるミニコンのALU部

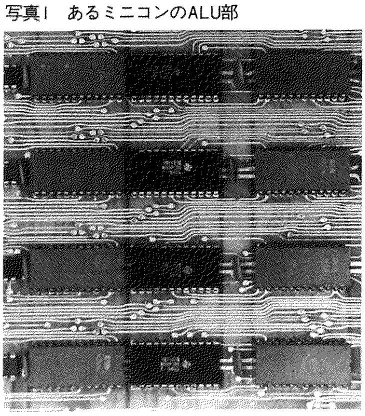


表1　SN74181の演算表（式はC言語に準じて表記しています）

<table>
<tr><th>M</th><th>S3</th><th>S2</th><th>S1</th><th>S0</th><th>~Cn = H (no carry)</th><th>~Cn = L (with carry)</th></tr>
<tr><td>L</td><td>L</td><td>L</td><td>L</td><td>L</td><td>F = A</td><td>F = A + 1</td></tr>
<tr><td>L</td><td>L</td><td>L</td><td>L</td><td>H</td><td>F = A | B</td><td>F = (A | B) + 1</td></tr>
<tr><td>L</td><td>L</td><td>L</td><td>H</td><td>L</td><td>F = A | ~B</td><td>F = (A | ~B) + 1</td></tr>
<tr><td>L</td><td>L</td><td>L</td><td>H</td><td>H</td><td>F = -1 (all 1)</td><td>F = 0</td></tr>
<tr><td>L</td><td>L</td><td>H</td><td>L</td><td>L</td><td>F = A + (A & ~B)</td><td>F = A + (A & ~B) + 1</td></tr>
<tr><td>L</td><td>L</td><td>H</td><td>L</td><td>H</td><td>F = (A | B) + (A & ~B)</td><td>F = (A | B) + (A & ~B) + 1</td></tr>
<tr><td>L</td><td>L</td><td>H</td><td>H</td><td>L</td><td>F = A - B -1</td><td>F = A - B</td></tr>
<tr><td>L</td><td>L</td><td>H</td><td>H</td><td>H</td><td>F = (A & ~B) - 1</td><td>F = A & ~B</td></tr>
<tr><td>L</td><td>H</td><td>L</td><td>L</td><td>L</td><td>F = A + (A & B)</td><td>F = A + (A & B) + 1</td></tr>
<tr><td>L</td><td>H</td><td>L</td><td>L</td><td>H</td><td>F = A + B</td><td>F = A + B + 1</td></tr>
<tr><td>L</td><td>H</td><td>L</td><td>H</td><td>L</td><td>F = (A | ~B) + (A & B)</td><td>F = (A | ~B) + (A & B) + 1</td></tr>
<tr><td>L</td><td>H</td><td>L</td><td>H</td><td>H</td><td>F = (A & B) - 1</td><td>F = A & B</td></tr>
<tr><td>L</td><td>H</td><td>H</td><td>L</td><td>L</td><td>F = A + A</td><td>F = A + A + 1</td></tr>
<tr><td>L</td><td>H</td><td>H</td><td>L</td><td>H</td><td>F = (A | B) + A</td><td>F = (A | B) + A + 1</td></tr>
<tr><td>L</td><td>H</td><td>H</td><td>H</td><td>L</td><td>F = (A | ~B) + A</td><td>F = (A | ~B) + A + 1</td></tr>
<tr><td>L</td><td>H</td><td>H</td><td>H</td><td>H</td><td>F = A - 1</td><td>F = A</td></tr>
<tr><td>H</td><td>L</td><td>L</td><td>L</td><td>L</td><td colspan="2">F = ~A</td></tr>
<tr><td>H</td><td>L</td><td>L</td><td>L</td><td>H</td><td colspan="2">F = ~(A | B)</td></tr>
<tr><td>H</td><td>L</td><td>L</td><td>H</td><td>L</td><td colspan="2">F = ~A & B</td></tr>
<tr><td>H</td><td>L</td><td>L</td><td>H</td><td>H</td><td colspan="2">F = 0</td></tr>
<tr><td>H</td><td>L</td><td>H</td><td>L</td><td>L</td><td colspan="2">F = ~(A & B)</td></tr>
<tr><td>H</td><td>L</td><td>H</td><td>L</td><td>H</td><td colspan="2">F = ~B</td></tr>
<tr><td>H</td><td>L</td><td>H</td><td>H</td><td>L</td><td colspan="2">F = A ^ B</td></tr>
<tr><td>H</td><td>L</td><td>H</td><td>H</td><td>H</td><td colspan="2">F = A & ~B</td></tr>
<tr><td>H</td><td>H</td><td>L</td><td>L</td><td>L</td><td colspan="2">F = ~A | B</td></tr>
<tr><td>H</td><td>H</td><td>L</td><td>L</td><td>H</td><td colspan="2">F = ~(A ^ B)</td></tr>
<tr><td>H</td><td>H</td><td>L</td><td>H</td><td>L</td><td colspan="2">F = B</td></tr>
<tr><td>H</td><td>H</td><td>L</td><td>H</td><td>H</td><td colspan="2">F = A & B</td></tr>
<tr><td>H</td><td>H</td><td>H</td><td>L</td><td>L</td><td colspan="2">F = 1</td></tr>
<tr><td>H</td><td>H</td><td>H</td><td>L</td><td>H</td><td colspan="2">F = A | ~B</td></tr>
<tr><td>H</td><td>H</td><td>H</td><td>H</td><td>L</td><td colspan="2">F = A | B</td></tr>
<tr><td>H</td><td>H</td><td>H</td><td>H</td><td>H</td><td colspan="2">F = A</td></tr>
</table>

よってはこの通り道の一部にビットをずらす（第0bitのデータを第1bitの位置へ，第1bitのデータを第2bitの位置へというように，あるいはその逆）配線を行っておいて，シフト演算を実現するものもありますが．それでも隣のビット線との関係しかありませんからALUのように楽に分割できます．

反面，命令解釈部や，タイミングの制御を行う部分は単純には分割できません．そもそもタイミング制御などはデータを扱う部分ではありませんから．この部分は，次回に解説するマイクロプログラムという手法で実現されることが多く，それなりにシステマティックにして回路の複雑さを押さえる工夫がされていました．

ユニット分割方法の見なおし

ビット・スライス型の登場

ビット・スライスの考え方の最初のアイディアは，いま見てきたように，各ユニットを少ないビット数単位に分割した単純なモジュールを並べることで実現することでした．次の1歩は，ユニットの分割のしかたを見直すことです．

そもそも，4bit単位などにモジュールを分割したのは，ICのパッケージの問題が大きかったのです．そもそも，当時は64ピンのパッケージなどは一般的ではなく，最近よく使われるさらに多ピンのパッケージはありませんでした．また，ICを用いる利点のひとつには外部の配線の量を減らすこともあります．多ピンのパッケージを使っても，外部に錯綜した複雑な配線が必要なら，回路全体としては小型化することはできません．つまり，ICのパッケージは小さいまま，したがってピン数は同じ程度にして，外部の配線の量を減らすための工夫をしなければなりません．

そのためには，外で必ず接続する相手がいて配線しなければならないのなら，その相手ごとひとつのチップに入れてしまえ，という考え方が有効です．つまり，ALUとレジスタをひとつのICに入れて，さらにデータの通り道の切り替えスイッチまで含ませてしまうのです．ただし，ひとつのパッケージに入れるビット数は少ないままにしておきます．こうすれば，データの入出力線はせいぜい処理するビット数の3倍程度ですから，少ないピン数で配線の量を大幅に減らせます．

このようなICを使うと，CPUの構成は図4のようになります．ほら，CPUを複数のICで実現するとき，機能ブロックごとに分割せずに，あるビット数の幅で複数のブロックを輪切りにしたように，言い換えればスライスしたように実現しているでしょう．これがビット・スライスの名の由来です．

ビット・スライス型ICの系譜

その発展と衰退

では，簡単に歴史を振り返ってみましょう．

1974年にインテル社から3002という8bit幅のビット・スライス用ICが出ています．でも，まだ，手探りの進化段階とでもいうべきものです．

1975年にはフェアチャイルド社から，MACRO LOGICシリーズ（写真2）というビット・スライス型マイクロプロセッサのシリーズが発表され，さらにAMD社から決定版ともいえるAM2901を筆頭とするAM2900シリーズが発表されています．以上3種のビット・スライス用のライン・アップは，TTLを中心とした高速のバイポーラ・プロセスを利用したものです．これらを使用すれば，当時のミニコンピュータ・クラスのCPUを比較的容易に実現できました．とくにヒット商品(?)となったAM2900シリーズによって，ビット・スライスの考え方が普及したようなところがあります．これらは，100nsから200nsで1演算を実行できる程度の速度です．クロック周波数にして，5MHzから10MHzといったところでしょうか．もっとも，これはつまるところCPU内部の部品の速度ですから，これらを使ったCPUの速度は割り引いて考える必要があります．

1976年にはモトローラ社がMC10800という，バイポーラ・プロセスの中でも最高速を誇るECLプロセスで，ビット・スライス型のチップを発表しています．このMC10800は，大型コンピュータのCPUにも応用されるほどの能力がありました．

1977年にはAM2901の後継として，AM2903というALU部分が発表されています．これは乗除算などの高度な演算をより容易に実現するための工夫などが取り入れられていました．

しかし，1980年代に入るころには衰退の道をたどります．16bitマイクロプロセッサが発表され，すでに下位ミニコンピュータ・クラスの能力はマイクロコンピュータの能力に陵駕されはじめてきました．また，高速化するためには，個別の部品からなる回路を使うより，多少遅いプロセスを用いてでもひとつのチップにしてしまったほうが有利なこともありました．ICの入出力端子につながっているトランジスタは，電流を多く流したり保護回路が入っているために，IC内部のトランジスタよりはるかに遅いからです．1チップにすれば遅いトランジスタの数を減らせます．したがって，より高速をめ

ざした上位ミニコンピュータも独自開発のLSIを採用するようになって，汎用をめざしたビット・スライス型のマイクロプロセッサは使われなくなってきました．また，ビット・スライス型マイクロプロセッサの応用分野のひとつであった，高速の入出力装置の制御（ディスク・コントローラやグラフィック描画プロセッサなど）についても，1チップのLSIが続々と開発されてきました．

そんなわけで，どうしてもビット・スライス型マイクロプロセッサを使う必要のあるアプリケーションが，徐々に少なくなってしまったのです．

ところで，今回は命令解釈と実行をつかさどる方法については述べませんでした．次回には，ビット・スライス型のマイクロプロセッサと深く関わりのあるマイクロプログラム方式と呼ばれる，命令解釈・実行の制御法について説明したいと思います．

写真2　フェアチャイルド社の
MACRO LOGICシリーズ

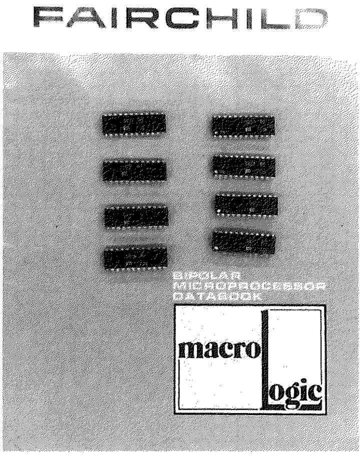
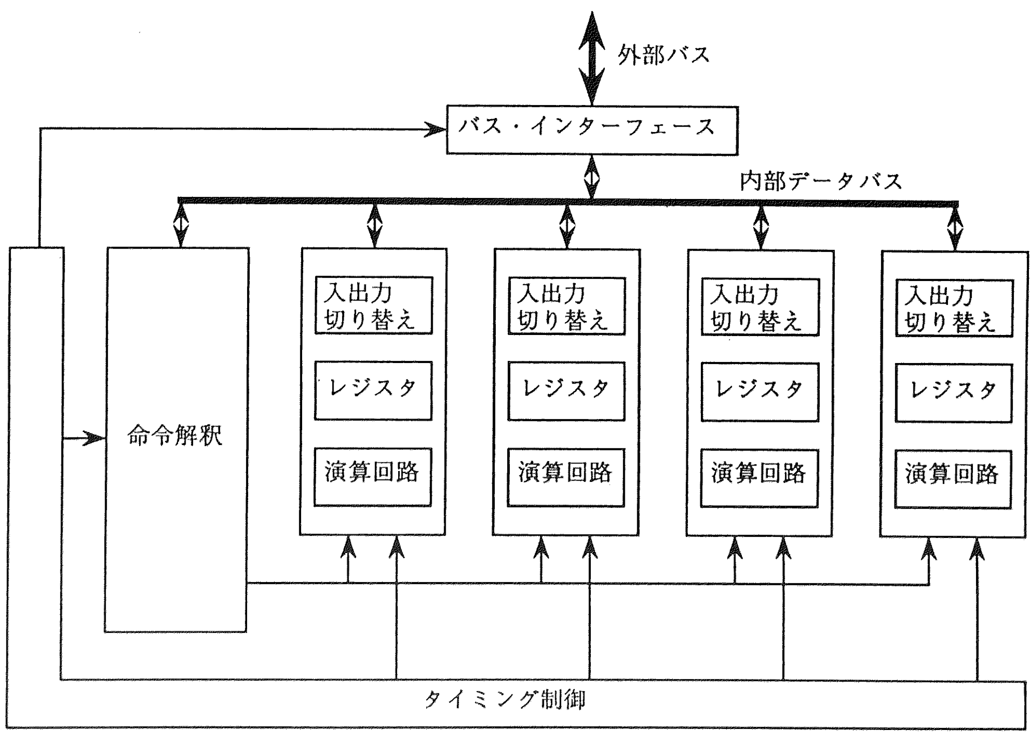


図4　ビット・スライス型マイクロプロセッサによるCPU構成の例



各演算ユニットでは4bit単位での処理を行うことで16bit単位の処理が行われる．

第2回

冬休みの構造化制御構文編

酒井法雄

先月は，イカすプログラミングのために何が必要であるかといった，一般論を述べました．その中でも，構造化という発想自体が重要であるということをお話ししました．

そこで今月は，構造化の中核をなす構造化制御構文とその使い方について解説します．

名義変更

突然ですが，私はクルマを買いました．クルマといっても，前回の最後の例にあった新型RX-7ではありません．なんと，ワンボックスのワゴンです．しかも，あと1回2年車検が取れるという中古車です．しかも，これが激安の5万円です．程度も悪くありませんし，半年以上車検がついています．回転対座シート付きで，8人も乗れます．フルフラットにすれば，しっかり寝ることもできます．個人売買はラッキーですね．

早速5万円を払ってクルマを引き取ってきました．幸い駐車場は確保できましたが，駐車場代が月に4万円ですから，クルマとたいして変わりません．それでも，探してもないのが都内の駐車場ですからよかったといえるでしょう．

しかし，ここからが苦難の始まりでした．名義変更をしなくてはならないのです．私は以前に原付オートバイを友人から譲り受け，自分で名義変更をしたことがあったので，たやすく考えていました．しかし，クルマではそうたやすくはありませんでした．

このクルマは，私の個人名義ではなく，友人とやっている会社で買ったものです．クルマの名義変更には印鑑証明が必要です．個人名義ならば区役所に行ってスグにとれるのですが，会社名義にしたので，会社の代表者の印鑑証明を法務局に行ってとってこなくてはいけません．法務局はいつ行っても大変混雑しています．オマケに出入りしている人種がどうも一般人ではありません．どうにも気遅れしてしまいました．結局，印鑑証明をとるだけで半日つぶれてしまいました．

次に，車庫証明です．警察署に行って必要書類をもらい，駐車場の持主に送って証明書を書いてもらいます．それを受けとってから警察署に申請を出します．それから数日しないと車庫証明はもらえません．結局，車庫証明をもらうまでに2週間かかりました．

クルマを譲ってくれた人からの書類とこちらで必要な書類が揃ったので，陸運局に行きました．しかし，ここでも苦難が待ち受けていました．法人名義にするための文例がなく，書類を書くのに難儀してしまいました．こんなことなら代書屋に頼めばよかったなどと考えながらも，あちこちの窓口を回ったあげく，やっと新しいナンバーになったときには，すでに夕方になっていました．

お役所での手続きは面倒なものです．何をするにも1か所では済みません．必ず収入印紙を売っているところとか，ナンバーを引き取るところとか，

あちこちを回らされます．クルマ屋さんやオートバイ屋さんは，こんなことを毎日しているのかと思うと，私はそんな商売でなくてよかったと思ってしまいました．

手続きと処理

お役所の手続きは何かと面倒なものですが，それが世の中のしくみですからしかたありません．よく考えてみると，このような手続きは私たちの日常にもたくさんあります．

私が面倒だと思う日常の手続きは，風呂と洗濯です．風呂は入るだけならまだいいのですが，私の家の風呂は貯め置き式なので，湯を沸かして湯舟に入れて適切な温度にしなくてはいけません．この作業が面倒で，なかなか風呂に入れません．湯を入れてからタイルが温まるまで待たなければいけないので，3回に1回は待っている間に寝てしまいます．

このように考えてくると，私たちの日常自体が手続きとその処理であるといえます．そういえば，小学生の夏休みや冬休みには，時刻表のように1日の生活パターンを決めて，書かされました．私はどうにもこの作業が嫌いでした．決めたところで，どうせそのとおりにできるわけはないのですから．

このように手続きを順に記述していくのは，CやBASICなどのプログラミングと同じです．これらの言語は，手続き型言語といわれ，やるべきことを順に記述していきます．コンサートなどの進行表をプログラムということを考えれば，コンピュータでいうプログラムという言葉は，そもそもはこのような手続きを記述することから発生しているということなのでしょう．

プログラミングとは，処理したいこと，処理すべきことを順に記述することにほかならないわけです．

QuickBASICの構造化制御構文

さて，そろそろ本題に入りましょう．今回のテーマは，構造化制御構文でした．この構造化制御構文の御利益とは何かというと，処理したい手続きをわかりやすく記述できるということです．プログラムは，基本的には書かれていることを順に実行していきます．しかし，必ずしも一本道ではありません．むしろ，いろいろな条件によって分岐したりループしたりします．このようなプログラムの流れの制御をするのが，制御構文です．

プログラミングでの制御構文には，大きく分けて条件分岐とループがあります．この2つの制御ができれば，通常のプログラミングはすべてできてしまいます．

条件分岐とは，ある条件に合致したときにすべき処理を指定するものです．たとえば，「便意をもよおしたらトイレに行く」というのは条件分岐です．条件分岐がないと，必ずある順番でその処理をしなくてはならなくなります．いつも同じ時刻にトイレに行くことになっていたら，腹をこわしたときにたいへんです．

ループは，繰り返しの処理です．「茶碗にご飯がなくなるまで飯を喰う」というのはループです．ループがなければ，同じ処理を何度も書かなくてはならなくなってしまいます．

では，実際にQuickBASICの構造化制御構文を紹介しましょう．ここでは，私の嫌いだった先ほどの冬休みの1日の予定表を例にしてみます．

IF文

IF文は，御存知のとおり「もしなんとかだったら……」という条件分岐の処理を実現するものです．昔のBASICでは，このIF文の処理は必ず1行で書かなくてはいけませんでした．そこで複雑な処理をしたいときには，GOTO文やGOSUB文を使って一度「よそ」に行って処理してこなくてはいけません．しかも，「よそ」というのが行番号やラベルであったため，どこだかわかりにくいものでした．

QuickBASICのIF文は，処理の内容を複数行にわたって記述できます．その範囲は，END IF文までの間と決まっています．次の例を見てください．

```
IF getup < 6 THEN
        着替える
        学校に行く
        ラジオ体操をする
        帰ってくる
END IF
```

処理内容は，ここでは実際のプログラムではなく日本語で書いてあります．

この例では，IFに合致する処理内容を書くとき，1段インデントをつけています．このようにインデントすれば，処理範囲を明確にできます．この範囲をブロックと呼びます．もしインデントをしないと，図1のようになってしまいます．

このように1行で書くときには，END IFはいりません．しかし，処理

の内容がわかりにくくなってしまいます．インデントを使って処理のレベルとそのブロックを明確にしたほうが，ずっとわかりやすくなります．

次に，IFの条件に合わなかったときの処理の例をあげます．

```
IF getup < 6 THEN
        着替える
        学校に行く
        ラジオ体操をする
        帰ってくる
ELSE
        そのまま寝続ける
        風邪をひいたふりをする
END IF
```

この例では，IFの条件に合ったときと，そうでなかったときの2通りの処理が書かれています．条件に合ったときの処理範囲はTHENの次行からELSEの前行までです．条件に合わなかったときの処理範囲は，ELSEの次の行からEND IFの前の行までです．

しかし，場合によってはもっと複雑な処理が必要になるでしょう．このようなときには，ELSEIF文を使います．

```
IF getup < 6 THEN
        着替える
        学校に行く
        ラジオ体操をする
        帰ってくる
ELSEIF getup < 6.2
        急いで着替える
        走って学校に行く
        ラジオ体操をする
        帰ってくる
        ヤクルトを飲む
ELSEIF getup < 6.4
        超急いで着替える
        自転車で学校に行く
        ラジオ体操をする
        帰ってくる
        ひと休みする
        ヤクルトを飲む
ELSE
        そのまま寝続ける
        風邪をひいたふりをする
END IF
```

このように，ELSEIF文の後ろには，次のELSEIF，またはELSE，またはEND IFまでの間に処理内容を書くことができます．

SELECT CASE文

IF文を使えば，ある事象について分岐してそれぞれの処理ができます．しかし，このような処理にはもっと適役な構文が用意されています．それが，SELECT CASE文です．この構文では，ひとつの変数の内容によって処理を分岐します．先ほどのIF文の例をSELECT CASE文に書き換えたのが，次の例です．

```
SELECT CASE getup
CASE IS < 6
        着替える
        学校に行く
        ラジオ体操をする
        帰ってくる
CASE 6 TO 6.2
        急いで着替える
        走って学校に行く
        ラジオ体操をする
        帰ってくる
        ヤクルトを飲む
CASE 6.2 TO 6.4
        超急いで着替える
        自転車で学校に行く
        ラジオ体操をする
        帰ってくる
        ひと休みする
        ヤクルトを飲む
CASE ELSE
        そのまま寝続ける
        風邪をひいたふりをする
END SELECT
```

SELECT CASE文の後ろにあるgetupという変数が，いまから評価する変数です．その後ろのCASE文のあとにその変数の範囲を記述します．その範囲に合致したときにその次の行からの処理をします．処理されるのは，次のCASE文かEND SELECT文までの間です．

さて，この例では少々複雑なことをしています．最初のCASE文は，「CASE IS 条件」という形になっています．この例では，「IF getup < 6 THEN」という条件と同じです．

次の「CASE 6 TO 6.2」は，「IF 6 <= getup AND getup <= 6.2」と同じ条件です．したがって，厳密には先ほどのIF文の例とは違い，6.2のときもこの条件に合致することになります．境界条件が違ってしまうことに注意してください．

▼図1　インデントをしないで1行に書いた例

```
IF getup < 6 THEN 着替えて，学校に行って，ラジオ体操をして，帰ってくる．
```

ワンラインNEWS
米DECは開発中の「アルファ」プロジェクトの概要を明らかにした．「アルファ」はRISC方式を採用した次世代機で，CPUの処理速度は現行モデルの約4倍を実現する．

次の「CASE 6.2 TO 6.4」は「IF 6.2 <= getup AND getup <= 6.4」となり，前のCASE文にあった6.2のときという条件が重なります．しかし，SELECT CASE文では前のほうから評価していき，最初に条件に合った範囲の処理だけをします．したがって，条件の範囲が重なっても後ろのほうは無視されます．

「CASE ELSE」では，それ以外のときという条件ですから，IF文の「ELSE」と同じであると考えられます．

SELECT CASE文は，END SELECTまでの範囲に有効です．この例では，すべてのCASE文に範囲がありましたが，「CASE 7」と書けば「IF getup = 7 THEN」というIF文と等価になります．

このように，SELECT CASE文はひとつの変数について分岐する処理を書くときには，たいへんスマートに記述することができます．各処理内容についてインデントを使ってブロックをわかりやすくすることは，ここでも同じです．QuickBASICのIF文はブロックごとにきれいに記述できますが，複数の条件による分岐を処理するときに有効なものです．ひとつの変数による分岐であれば，SELECT CASE文を使ったほうがいいでしょう．

FOR～NEXT文

ループ処理をするとき，その回数がわかっているときに使われるのがFOR～NEXT文です．

```
FOR ch = 1 TO 12
        chチャンネルに切り換える
        テレビを見る
NEXT ch
```

この例は，VHFのテレビのチューナーを切り換えて，何の番組をやっているかをざっと見るといったことをするものです．チャンネルは1から12までと決まっていますから，このようにループカウントを指定できるわけです．

もし，奇数チャンネルだけを調べるのであれば，次のように指定できます．

```
FOR ch = 1 TO 12 STEP 2
        chチャンネルに切り換える
        テレビを見る
NEXT ch
```

この例では，1，3，5，7，9，11のチャンネルを見ることになります．

ここまでは，今までのBASICでも同じですが，QuickBASICには，「EXIT FOR」という文が用意されています．これは，FOR～NEXTループから脱出する文です．

```
FOR ch = 1 TO 12
        chチャンネルに切り換える
        テレビを見る
        IF 面白い番組 THEN
                EXIT FOR
        END IF
NEXT ch
```

ここでは，面白い番組をやっていたらという条件でEXIT FOR文を実行しています．つまり，面白い番組をやっていたら，チャンネルをスキャンすることをやめてそのチャンネルの番組を見ようということになります．もちろんGOTO文を使ってループを脱出することもできますが，EXIT FOR文を使えば脱出先は必ずNEXT文の次の行です．このほうがどこかに処理が飛ばない分，制御構造がわかりやすくなるわけです．

DO～LOOP文

ある条件によってループするときには，LOOP文を使います．この構文は，DO文ではじまり，LOOP文で終了します．その間の文を繰り返して実行します．

```
DO WHILE こづかいがある
        ライダースナックを買う
        カードをもらう
LOOP
```

この例では，こづかいがある限り，ライダースナックを買ってカードをもらうというものです．WHILEは，その後ろの文が「真である限り」という条件になります．真というのは，0以外の数値と等価ですから，「DO WHILE 1」は，無限ループになります．また，この例は次のように書くこともできます．

```
DO UNTIL こづかいがなくなる
        ライダースナックを買う
        カードをもらう
LOOP
```

ここでは，WHILEの代わりにUNTILが使われています．これは，その後ろの文が「真になるまで」という条件になります．したがって，WHILEのときと反対を意味する文をつければ，等価なループになるわけです．どちらを使っても同じことができるわけ

ですから，どちらかひとつがあればいいようなものですが，設定したい条件を明確にすることができるメリットがあります．

DO～LOOP文での条件は，ここまでの例ではループの頭で設定しましたが，ループの最後に設定することもできます．

```
DO
        ライダースナックを買う
        カードをもらう
LOOP WHILE こづかいがある
```

これは，先ほどの例と同じように見えますが，実は違います．というのは，こづかいがあるかを判断するのがループの最後ですから，もし，最初からこづかいがなくてもライダースナックを買ってしまうことになるのです．これでは駄菓子屋のオバサンに怒られてしまいますから，この構文を使うときには注意が必要です．条件に合おうが合うまいが，とにかく1度はループの内容を実行したいときにだけ使うべきです．たとえば，次のような例にはこの構文がいいでしょう．

```
DO
        宿題をする
LOOP UNTIL 根性がなくなる
```

これは冗談として，実際には次のようなときに使うと有効です．

```
DO
      LOCATE 10, 5
      INPUT "Select(1-5): ", sel
LOOP UNTIL 1 <= sel AND sel <= 5
```

この例は，1から5までの数値を入力してほしいとき，それ以外の数値や文字列などを入力されたときには再入力されるというものです．

ところで，ループの条件以外の条件で，ループを脱出したいときがあります．このようなときには，EXIT DO文を使います．次の例を見てください．

```
DO UNTIL こづかいがなくなる
     ライダースナックを買う
     カードをもらう
     IF ライダーキックのカード THEN
              EXIT DO
      ENDIF
LOOP
```

これならば，欲しかったライダーキックをしているカードが手に入れば，それ以上ライダースナックを買わなくても済むわけです．

御存知ない方のために書いておきますと，ライダースナックとは，仮面ライダースナックのことです．私が小学生のころには仮面ライダーが一世を風靡しました．ライダースナックを買うと，仮面ライダーの雄姿や怪人を印刷したカードがもらえました．このカード集めに全国の小中学生は夢中になり，カードだけを集めてスナックは捨てるなどということが相継ぎ，社会問題となりました．もっとも私はキカイダーのファンだったので，この遊びには無縁でした．

このほかにも，従来からあったWHILE～WEND構文などのループ制御構文がありますが，DO～LOOP構文だけで十分でしょう．

サンプルプログラム

ここまで，構造化制御構文の例をあげましたが，概念的なものでした．これはイカすというモノではありませんので，ここで実際に構造化制御構文を使ったプログラミング例を紹介しましょう．

ここで作ったプログラムは，ラケットゲームのプログラムです．コンピュータプレイヤーとユーザーでボールを打ち合って対戦します．ボールはラケットに当たるか左右の壁に当たると反射します．各プレイヤーは5コのボールをもっています．ボールを後ろにそらすとボールを失います．全部失ったら負けです．

テキストだけを使ったゲームですから，動きはあまりスムーズではありませんが，それなりに楽しめます．問題なのは，移動量が必ず1ですとゲームが単調になってしまうことです．そこで，ラケットの中央にボールが当たったときには打ち返せないスーパーショット，ラケットの端にボールが当たったときにはX方向に移動量が大きいヘンテコショットを打つことにしました．

また，[F]と[S]のキーでウェイトの量を変えてゲームの速度を変えられることにしました．

では，このゲームを構造化日本語で考えてみましょう．すると，図2のように書くことができます．

構造化日本語にすると，やるべき処理とその構造が明確になります．この構造化日本語にそってQuickBASICの文法でプログラムを書くと，リスト1（racket.bas）のようなプログラムになります．

いかがですか？　それぞれの対応を見れば，処理内容がわかります．構造化制御構文を使わないと，ピョンピョンとあちこちを飛び回るプログラムになってしまいます．この例でも，構造

化した成果は十分にあるようです.

しかし，どうもプログラムがだらだらと長くて，構造化日本語にくらべると処理内容がわかりにくくなっています．これでは，さほどカッコよくありません．構造化制御構文だけでは，どうしても全体が長くなってしまい，見通しが悪くなってしまうのです．また，似たような処理をあちこちに書いてあるのもいただけません．

こんなときには……そうです，サブルーチンでした．しかし，Quick-BASICには前回述べたようにプロシージャという概念があります．では，次回はそのプロシージャについて解説します．もちろんこのプログラムも，プロシージャを使ってさらにきれいに書き直すことにしましょう．

それでは，また来月もこのコーナーでお会いしましょう．

▼図2　ラケットゲームの内容

```
定数の初期化
変数の宣言
変数の初期化
DO
        面の初期化
        ボール数などの表示
        リターンキーでゲーム開始
        DO UNTIL  ボール位置がラケットの後ろになったら
                ウェイト処理
                ボールを消去
                IF コンピュータのラケットに当ったら
                        Y方向反転
                        ショット決定
                IF プレイヤーのラケットに当ったら
                        Y方向反転
                        ショット決定
                IF 両側の壁に当ったら
                        X方向反転
                IF ボールがX方向の画面範囲外ならば
                        壁位置に矯正
                IF ボールがY方向の画面範囲外ならば
                        ラケット位置に矯正
                コンピュータラケットの消去
                コンピュータラケットの新しい位置計算
                SELECT CASE ユーザーのキー入力
                CASE "4"
                        プレイヤーラケット消去
                        プレイヤーラケット左移動計算
                CASE "6"
                        プレイヤーラケット消去
                        プレイヤーラケット右移動計算
                CASE "S"
                        ウェイトの増加
                CASE "F"
                        ウェイトの減少
                END SELECT
                キーバッファ消去
                ボール描画
                プレイヤーラケット描画
                コンピュータラケット描画
        LOOP
        打ち損じた方のボールの数を減らす
        ボールロストのベル
LOOP UNTIL  いずれかのボールがなくなる
勝者を表示
```

```
'*****************************************************
'            Racket Game version 1.0
'         Copyright(c) Norio Sakai 1991
'            All Rights Reserved.
'*****************************************************
CONST BALL$ = "●"                               'ボール
CONST PAD$ = "========"                         'ラケット
CONST XL = 1                                    'コート左座標
CONST XR = 80                                   'コート右座標
CONST YT = 2                                    'コート上座標
CONST YB = 24                                   'コート下座標
CONST PY = YB                                   'プレイヤーラケットY座標
CONST DF = 500                                  '速度変化単位
CONST XS1 = 10                                  '速度表示X座標
CONST XS2 = 60                                  'ボール数表示X座標
CONST YSP = 25                                  'プレイヤー得点表示Y座標
CONST YSC = 1                                   'コンピュータ得点表示Y座標
CONST CY = 2                                    'コンピュータラケットY座標
CONST MX = 30                                   'メッセージ表示X座標
CONST MY = 12                                   'メッセージ表示Y座標
CONST MESSAGE$ = "Hit [Return] to Start"メッセージ

DIM px AS INTEGER, cx AS INTEGER                'ラケットX座標
DIM x AS INTEGER, y AS INTEGER                  'ボール座標
DIM dx AS INTEGER, dy AS INTEGER                'ボール移動単位
DIM i AS INTEGER, w AS INTEGER                  'ワーク、ウェイト
DIM pdx AS INTEGER, cdx AS INTEGER              'ラケット移動単位
DIM pb AS INTEGER, cb AS INTEGER                'ボール数
DIM sx AS INTEGER, sy AS INTEGER                'ボール移動係数

'**** 各種変数初期化 ****
dx = 1
dy = 1
w = 10000
pdx = INT(LEN(PAD$) / 2) - 1
cdx = pdx
pb = 5
cb = 5

DO        '**** メインループ ****
    CLS   '**** 面初期化 ****
    x = 40:y = 7
    px = 40 - (LEN(PAD$)):cx = px
    sx = 1:sy = 1
    LOCATE YSP, XS1:PRINT USING "Speed #####"; w;
    LOCATE YSC, XS2:PRINT USING "Balls :######"; cb;
    LOCATE YSP, XS2:PRINT USING "Balls :######"; pb;
    LOCATE MY, MX:PRINT MESSAGE$
    DO UNTIL INKEY$ = CHR$(13): LOOP
    LOCATE MY, MX:PRINT SPC(LEN(MESSAGE$));

   '**** 1ゲームループ ****
    DO UNTIL y < YT OR YB < y       'ボールがはみ出たら次の面
        FOR i = 0 TO w: NEXT i      'ウェイト
        LOCATE y, x:PRINT "  ";     'ボールの消去
```

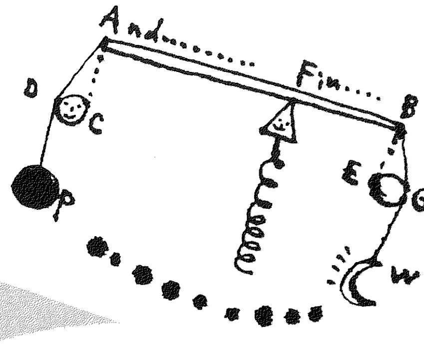
LIST 1

racket.bas

```
'**** コンピュータプレイヤー当たり判定 ****
IF (y <= YT AND cx <= x AND x < cx + LEN(PAD$)) THEN
    BEEP
    dy = -dy
    'スーパーショット
    IF x = cx + INT(LEN(PAD$) / 2) THEN
        sx = 2:sy = 4
    'ヘンテコショット
    ELSEIF x = cx OR x = cx + LEN(PAD$) - 1 THEN
        sx = 2:sy = 1
    'ノーマルショット
    ELSE
        sx = 1:sy = 1
    END IF
END IF
'**** プレイヤー当たり判定 ****
IF (y >= YB AND px <= x AND x < px + LEN(PAD$)) THEN
    BEEP
    dy = -dy
    'スーパーショット
    IF x = px + INT(LEN(PAD$) / 2) THEN
        sx = 2:sy = 4
    'ヘンテコショット
    ELSEIF x = px OR x = px + LEN(PAD$) - 1 THEN
        sx = 2:sy = 1
    'ノーマルショット
    ELSE
        sx = 1:sy = 1
    END IF
END IF
IF x <= XL OR XR <= x THEN  '両端の壁当たり判定
    dx = -dx
END IF
   x = x + (dx * sx)         '両端の壁当たりボール位置矯正
IF x < XL THEN
   x = XL
ELSEIF XR < x THEN
      x = XR
END IF
'**** 両プレイヤー当たりボール位置矯正 ****
y = y + (dy * sy)
IF y < YT THEN
    y = YT - 1
ELSEIF YB < y THEN
    y = YB + 1
END IF
'**** コンピュータラケット移動 ****
IF x < cx THEN                 'ボールが左にあるとき
    LOCATE CY, cx:PRINT SPACE$(LEN(PAD$));
    cx = cx - cdx
    IF cx < 1 THEN
        cx = 1
    END IF
ELSEIF cx + LEN(PAD$) < x THEN'ボールが右にあるとき
```

```
            LOCATE CY, cx:PRINT SPACE$(LEN(PAD$));
            cx = cx + cdx
            IF 80 - LEN(PAD$) < cx THEN
                cx = 80 - LEN(PAD$)
            END IF
        END IF
        '**** キー入力処理 ****
        SELECT CASE UCASE$(INKEY$)
        CASE "4"                            'ラケット左移動
            LOCATE PY, px:PRINT SPACE$(LEN(PAD$));
            px = px - pdx
            IF px < 1 THEN
                px = 1
            END IF
        CASE "6"                            'ラケット右移動
            LOCATE PY, px:PRINT SPACE$(LEN(PAD$));
            px = px + pdx
            IF 80 - LEN(PAD$) < px THEN
                px = 80 - LEN(PAD$)
            END IF
        CASE "S"                            '速度を遅くする
            IF w < 32767 - DF THEN
                w = w + DF
                LOCATE YSP, XS1 + 6:PRINT USING "#####"; w;
            END IF
        CASE "F"                            '速度を速くする
            IF 1000 < w THEN
                w = w - DF
                LOCATE YSP, XS1 + 6:PRINT USING "#####"; w;
            END IF
        END SELECT
        DO UNTIL INKEY$ = "": LOOP          'キーバッファ消去
        LOCATE y, x:PRINT BALL$;            'ボール描画
        LOCATE PY, px:PRINT PAD$;           'プレイヤーラケット描画
        LOCATE CY, cx:PRINT PAD$;           'コンピュータラケット描画
    LOOP
    '**** ボールの数を減らす ****
    IF y < YT THEN
        cb = cb - 1
    ELSE
        pb = pb - 1
    END IF
    FOR i = 0 TO 10:BEEP:NEXT i         'ボールロストのベル
LOOP UNTIL cb = 0 OR pb = 0             'いずれかのボールがなくなるまでループ
'***** 勝ち判定 ****
LOCATE MY, MX
IF cb = 0 THEN
    PRINT "You Win!"
ELSE
    PRINT "Computer Win!"
END IF
END
```

Illustration : Masami Nozue



アルゴリズム篇　第2回
ソート(並べ替え)・その2

実践C塾

三代目塾頭
㈱ジャステックラボ
加野島英渡／山本博昭

先月は，並べ替え（ソート）の代表的なアルゴリズムとして「選択ソート」と「バブルソート」を紹介した．今回は，さらに「ラディックスソート」，「シェルソート」，「クイックソート」のアルゴリズムを解説する．また，前回作成したWCOUNT.EXEをクイックソートを用いて高速化してみる．

C1 先月までのあらすじ

データ処理のための代表的なアルゴリズムを紹介しながら，C言語で実際にツールを書いてみるという趣旨でスタートしたこの連載だが，先月は初回ということでテーマツールの仕様を決めてそれを作るということにページの大半を割いてしまった．そのため，肝心のアルゴリズムのほうは，ツールのテーマである「単語の出現頻度」をカウントした結果を表示するために使うデータの並べ替えの手法として「選択法」,「バブルソート」という2つのソートを紹介したところで原稿枚数がいっぱいになり，少々ものたりなさを残してしまったようである．テーマツールのほうは，基本仕様にしたがって一応動作するものができあがったが，単語のABC順一覧表示にバブルソートを使用しているところなど，もっとほかのアルゴリズムを使って高速化する余地がありそうである．

C2 今月の重点項目

先月号でテーマツールの一覧表示部分に使用した「バブルソート」は，ロジックもシンプルでプログラムが書きやすいアルゴリズムだが，文字列をABC順に並べ替えるというような処理においては，決して高速なものとはいえない．小さなファイルならともかく，10KB以上のファイルを処理させると，画面表示がしばらく停止状態になることがよくわかる.「アルゴリズムの宝庫」といわれるソートの中には，もっとこういった処理に適した高速なものがある．

というわけで,前回に引き続いて,ソートのアルゴリズムをいくつか紹介する．その中からバブルソートより効率のよいものを選んで，テーマツールのソート部分の高速化をはかることにしよう．アルゴリズムを変更することによって，果たしてどの程度の効果があるものなのか,興味深いところである．

C3 ソートいろいろその2

【ラディックスソート】

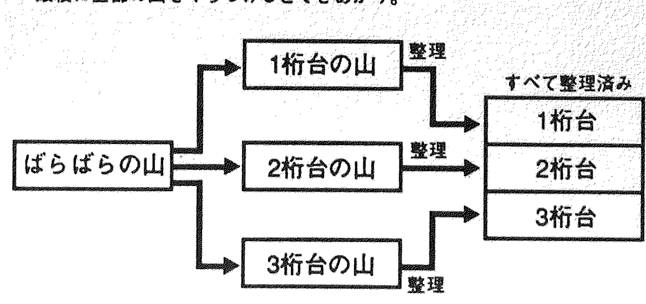
先月号で，子供に数字を書いたカードを並べ替えさせる話を書いたが，1から10くらいまでならともかく，100までとか1000までとかのカードとなると先月紹介した「選択ソート」を用いていたのでは大騒ぎになる．ここで大人なら少し知恵をしぼって図1のような方法を考えつくかもしれない．

巨大なカードの山の中から1を探して，2を探して……というのよりは効率がよさそうである．よさそうではあるが，実際にやってみることを考えるといまひとつ割り切れないものを感じるのは,「1桁の山ならともかく，3桁の山というのは元々のばらばらの山とそう変わらない分量があるのではないか？」という素朴な疑問である．いいところに気がついた．このやり方の問題点は，せっかく分けたおのおのの山の分量が極端に違うことである．これを解決するためには，たとえば2桁台の山であれば10番台の山，20番台の山というようにさらに細かく分けていくことになる．

さらに問題はもう1つある．今までの例は，1から1000までの連続した数字をそろえるということであったが，とびとびになっているような数値を分類しようとするときには「山」をいくつ作ればいいのか予想がつかない．こ

図1

・まずカードを1桁台、2桁台、3桁台などとおおまかに分けて山にしておく。
・その山をそれぞれ並べ替えていく。
・最後に全部の山をくっつけるとできあがり。



れはコンピュータのプログラムにとってはかなりイヤな話である．プログラムで「山」を作るということは，なにがしかの領域を使うということであるが，どのくらいその領域を使うのかわからないプログラムというのはむずかしいものである．

では，この分類法をもうちょっと別の視点から考え直してみよう．問題点は

・おのおのの山に積まれるデータの数が極端に違ってくる
・いくつ山を作ればいいのか予想がつかない

ということである．前置きがずいぶんと長くなってしまったが，ラディックスソートの基本的な考えがこの問題をクリアしてくれる．図2を見てほしい．「元々の山を末尾の桁から分類していく」というのがポイントである．これならどんなデータがいくつこようと山の数は最大10個である．ただし，ラディックスソートはあらかじめ最大の桁がわかっていないと使用することができない．末尾の桁から分類していって，どこまで上の桁まで分類を続けるのか判断がつかないからである．ラディックスソートは，まったくランダムな値が出現するデータには不適だが，長さが固定の文字列の並べ替えなどには威力を発揮する．

【シェルソート】

データ列を並べ替えするときに，前もっておおざっぱに分類しておいて，最後に仕上げをしたほうが効率がいいことがだんだんわかってきた．これから説明する「シェルソート」は，まさにその手法である．最初は間隔を粗くとって分類していき，だんだんと細かくしていく．間隔が1になったときの処理が終ればソートが完了していることになる．

具体的なデータを使ってシェルソートの流れを追っていこう．図3の①がソート前のデータ状態である．15個のデータがばらばらに並んでいる．これをまず②では7つおきにソートを開始する．15個のデータを7つおきに見にいくのであるから，最大でも3つのデータ同士の比較となる．左端から7つおきのデータをソートし終わったら，今度は右に1つずれて，同じように順次ソートしていく．1つずつ右にずらしながらすべてのデータを7つおきに見ていったら1度目の処理が終了する．

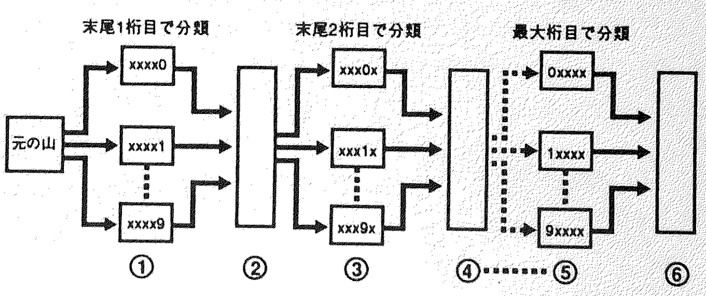
2回目の処理は，③から始まる．今度は間隔が1つ減って6つおきのソートとなる．2回目の処理も6つおきということを除けば，1回目で行った②からの処理と同一である．以降，④から5つおきの処理，⑤から4つおきの処理，⑥，⑦と少々くどいようだがソートの進行状態を知るために見ていってほしい．最後の仕上げは⑧である．ここでの処理の間隔は1，つまり左端からすぐ右隣のデータとの比較をして必要なら入れ替える処理を行っていく．右端まで並べ替え終わったら，ソートが完了していることになる．

ところで，ある間隔でソートしているときに「1，9，19，8」というようなデータの並びが出てきたとしよう．左端から見ていくと，「19」と「8」の時点ではじめて入れ替えが発生する．ところがこの2つのデータを入れ替えると，結果として今度は「9」と「8」が逆になってしまう．シェルソートの場合，その部分内のソートは完全に終わっていなければならないので，

「1度入れ替えがおきたら，1つ左に遡って逆順になっていないか再確認する」

という処理が必要になってくる．ここで入れ替えが必要になったらまた1つ

図2　ラディックスソートの基本的な考え方



①まずカードを末尾1桁で分類する。
②それぞれの山を積み上げる。
③今度は末尾2桁目で分類する。
④再び1つの山に積み上げる。
　・以降、1つずつ桁をあげて分類 ─→ 積み上げを繰り返す。
⑤最大桁を分類する。
⑥それを積み上げたら処理完了。

左に戻って同様の処理を行う（図4）．

シェルソートは，データ列を全体的におおまかにソートしはじめて次第に完成に近づけていくという特徴がある．ちなみに「シェル」という名は，「貝がら」やガソリンスタンドとは何の関係もなく，この手法を提唱した人の名前だそうだ．

リスト2はシェルソートのアルゴリズムを使ったサンプルプログラム「SSORT.C」である．例によってBorland C++でコーディングしてあるが，ほとんどのCコンパイラでそのままコンパイルできるはずである．前述のように，シェルソートはある間隔からとびとびにソートをはじめて，次第にその間隔を狭くしていく．間隔が1のときのソートが終わった時点で処理完了となる．「SSORT.C」の場合は，間隔をデータ総数の1／2からはじめて，それを半分にしながら間隔1までのソートを行っている．

では，コーディングを具体的に見ていくことにしよう．図5にSSORT.Cのソート実行部分の解説をまとめる．

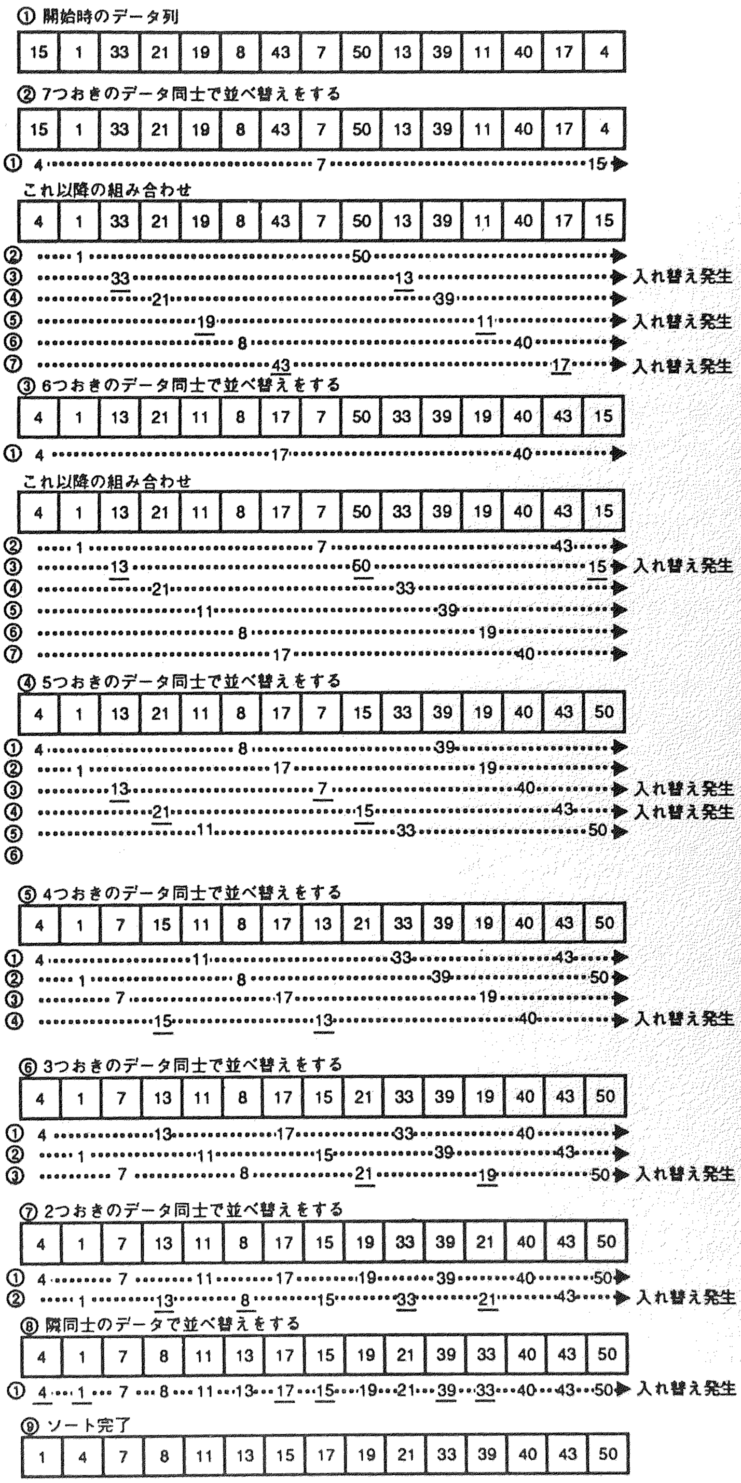
図6がシェルソートによるサンプルプログラム「SSORT.EXE」の実行画面である．各間隔で入れ替えが発生する様子を確認してほしい．また，ソース冒頭に元データ配列のコーディングがあるので，いろいろなデータをシェルソートにかけてみるのもよいだろう．

【クイックソート】

ソート紹介のトリをかざるのは，いかにも速そうな名前の「クイックソート」である．「クイックソート」の基本的な考え方は「データ列をある値より大きいものと小さいものにふり分け続けていく」というものである（図7）．

ふり分けを続けていって，すべての

図3 シェルソートの方法



データ列のデータ数が1になればソートは完了である．実際の数値をクイックソートを使ってふり分けていく過程を示したのが図8である．ふり分ける判定条件となる値nを中央値とよぶことにして，任意に決めた中央値より小さいものを左側に，大きいものを右側にふり分けていく．分けられた2つの各ブロックに対して新しい中央値を決めて再びふり分けを行い，ブロックの大きさが1になるまで続ける．

基本的な考え方はわかっていただけたであろうか．次にこのアルゴリズムを実際のプログラムでどう実現していくかを考えることにする．まず問題なのは「中央値」をいかにして決定するかである．前述の説明では説明に具合がいいように任意に決めていったが，プログラムではこういうことはできない．クイックソートをプログラムするときは，中央値を実際のデータの中から1つ選ぶことになる．具体的に書くと，図9のようなデータ列があるとする．これをクイックソートするためには次のような手順にする．

①中心値を配列先頭からとることにして，「5」を選択する．

②配列の先頭から左側ポインタを移動しながら中心値以上の値を探す．図10の場合，5が見つかる．

③配列の末尾から右側ポインタを移動しながら中心値より小さな値を探す．図10の場合，2が見つかる．

④左右のポインタが交差または一致していないので，両者を交換する．

⑤再び左側のポインタを配列末尾へ向かって進める．図11の場合，8が見つかる．

⑥右側のポインタも配列先頭へ向かって進める．図11の場合，4が見つかる．

⑦この場合，左右のポインタが交差しているので，これ以上の交換はない．

⑧この時点で，左端から右側ポインタの位置までは4以下の値が入っている．

⑨おなじく，左側ポインタから右端までは5以上の値が入っている．

⑩それぞれのブロックに対して，①～⑦と同様の処理を行う．
新しいブロックの中央値は，左側が2，右側が8である．

⑪ブロックが細分化されていって，大きさが1になれば，そのブロックは処理終了である．すべてのブロックのサイズが1になったところで，ソートが完了している．

⑩以降が少しわかりにくかったかもしれない．「同様の処理を行う」と，はしょってしまったが，実際のプログラムではこういうときに自分自身を呼び出すことになる．いわゆる再帰呼び出しである．このへんは，実際にこのソート手法をCでコーディングしたものをみていただくほうがいいだろう．図13がクイックソートのアルゴリズムによるソートの実例である．この関数の最後のほうで自分自身を呼び出している．自分によって呼び出された自分は，ブロックの大きさが1になるまでまた自分を呼び出して……というふうに繰り返していく．

リスト4は，この関数を組み込んだ

図4

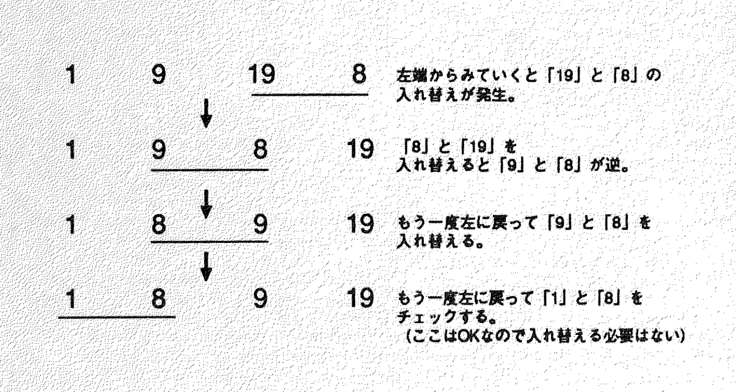


図5 シェルソートの実際のコーディング

```
:       for(soto_int = DSIZE / 2 ; soto_int >= 1 ; soto_int /= 2) {
 変数soto_intが，現在のソート間隔である．初期値は全データ数の1/2としている．処理は，ソート間
隔を1/2しながら1までくりかえす．

:           for(uchi_int = 0 ; uchi_int < DSIZE - soto_int ; uchi_int++) {
 あるソート間隔で左端から1つずつポインタuchi_intをずらしながら全データをソートしていく．

:               for(left = uchi_int;left < DSIZE - soto_int ; left += soto_int)
 とびとびに右端までソートしていくループ．変数leftは，左側のデータのポインタとなる．

:                   right = left + soto_int ;
 データの右側は，左側データからソート間隔分隔たった位置である．

:                   if(base_data[left] > base_data[right])  {
:                       swap_data(left , right);
 比較の結果，入れ替えが必要ならswap_dataを実行する．
```

図6 SSORT.EXE実行画面

```
A>SSORT
元データ     8  1  3  9  4  5  6  2 10  7
間隔 : 5     5  1  3  9  4  8  6  2 10  7
間隔 : 5     5  1  2  9  4  8  6  3 10  7
間隔 : 2     2  1  5  9  4  8  6  3 10  7
間隔 : 2     2  1  4  9  5  8  6  3 10  7
間隔 : 2     2  1  4  8  5  9  6  3 10  7
間隔 : 2     2  1  4  8  5  3  6  9 10  7
間隔 : 2     2  1  4  8  5  3  6  7 10  9
間隔 : 2     2  1  4  3  5  8  6  7 10  9
間隔 : 2     2  1  4  3  5  7  6  8 10  9
間隔 : 1     1  2  4  3  5  7  6  8 10  9
間隔 : 1     1  2  3  4  5  7  6  8 10  9
間隔 : 1     1  2  3  4  5  6  7  8 10  9
間隔 : 1     1  2  3  4  5  6  7  8  9 10
```

リスト1　SSORT.MAK

```
.AUTODEPEND

#        *Translator Definitions*
CC = bcc +SSORT.CFG
TASM = TASM
TLINK = tlink

#        *Implicit Rules*
.c.obj:
  $(CC) -c {$< }

.cpp.obj:
  $(CC) -c {$< }

#        *List Macros*

EXE_dependencies =  ¥
  SSORT.obj

#        *Explicit Rules*
SSORT.exe: SSORT.cfg $(EXE_dependencies)
  $(TLINK) /x/n/c/P-/LA:¥BC¥LIB @&&|
c0s.obj+
SSORT.obj
SSORT
        # no map file
emu.lib+
maths.lib+
cs.lib
|

#        *Individual File Dependencies*
SSORT.obj: SSORT.c

#        *Compiler Configuration File*
SSORT.cfg: SSORT.mak
  copy &&|
-ff-
-vi
-IA:¥BC¥INCLUDE
-LA:¥BC¥LIB
-P-.C
| SSORT.cfg
```

リスト2　SSORT.C

```
/* ================================================================

    SSORT.C        Copyright 1991 by JUSTECH-LABO. (H.kanoshima)

=============================================================== */
#define DSIZE   10
#define DLEFT   0
#define DRIGHT  (DLEFT + DSIZE - 1)

int     base_data[DSIZE] = { 8, 1 , 3 , 9 , 4 , 5 , 6 , 2 , 10 , 7};

/* ============================================================== */

void    dump_data(char *guide)
{
int     i;

    printf("%s",guide);
    for(i = DLEFT ; i < DSIZE ; i++)
        printf("%4d",base_data[i]);
    puts("");
}

/* ============================================================== */

void    swap_data(int x , int y)
{
int     tmp;

    tmp             = base_data[x];
    base_data[x]    = base_data[y];
    base_data[y] = tmp;
}

/* ============================================================== */

void    ssort(void)
{
int     soto_int , uchi_int;
int     left , right;

    for(soto_int = DSIZE / 2 ; soto_int >= 1 ; soto_int /= 2) {
        for(uchi_int = 0 ; uchi_int < DSIZE - soto_int ; uchi_int++) {
            for(left = uchi_int;left < DSIZE - soto_int ; left += soto_int) {
                right = left + soto_int ;
                if(base_data[left] > base_data[right])  {
                    swap_data(left , right);
                    printf("間隔 : %d ",soto_int);
                    dump_data(" ");
                }
            }
        }
    }
}

/* ============================================================== */

void    main(void)
{
    dump_data("元データ  ");
    ssort();
}
```

図7

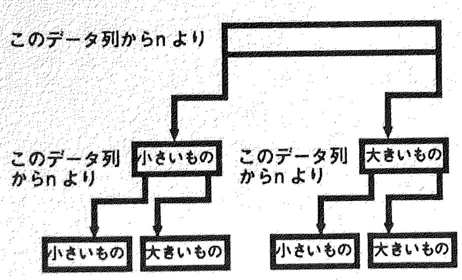


図8　クイックソートの基本的な考え方

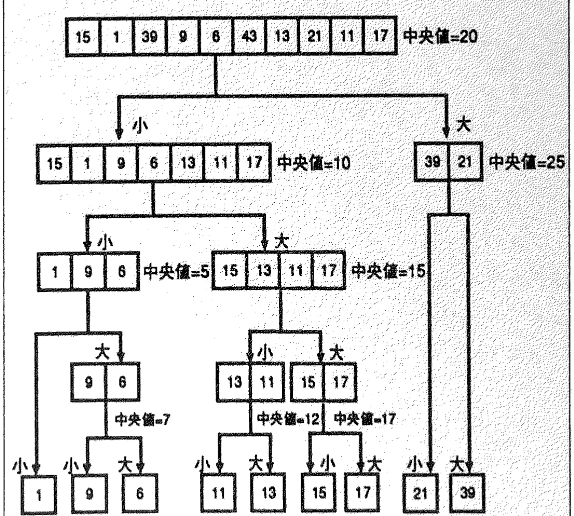
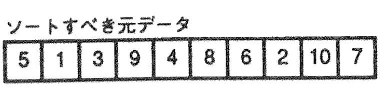


図9



テスト用プログラム「QSORT.C」である．このプログラムは，配列として定義された数値をクイックソートを用いてソートしながら入れ替えがおきるたびにその途中経過を表示する．シェルソートサンプルプログラム「SSORT.EXE」と同様に，起動は「A>QSORT」と行う．

図14がこのプログラムの実行結果の画面であるが，さきほどの例と同じ数値をコーディングしてあるので，ブロック分けがどのように進行しているか，実際に画面で確認していただきたい．

クイックソートは，データがランダムに出現するときに非常に効率のよい手法である．反面，データがすでにきちんと並んでいる部分があるような場合には，ブロックの分割に偏りが出てきて効率が悪くなる．ブロックの分割の効率は中央値の値によって左右されるわけで，理論的にはそのブロックに含まれる全データの平均値が望ましい．だからといっていちいちその平均値を計算している時間があったら，ソートをやっていたほうが早いというのが現実である．ランダムなデータ列の場合，ある時点でブロックに偏りが出てもそれを補って余りあるほどクイックソートの効率はよい．そのため，今回のプログラムでは，各ブロックの先頭データを中央値として使うことにして中央値を得るためのアドレスの計算を省略できるようにした．

また，さきほどの「QSORT.C」では，ブロック長が1になるまでクイックソートのアルゴリズムでソートしているが，たとえばブロック長が2のときは両方のデータを単純に比較したほうが高速である．これは，あとでWCOUNT.EXEの改造をするときに実際にやってみることにする．その

図10

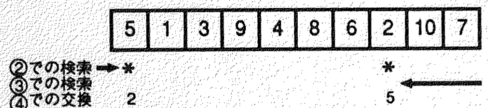


図11

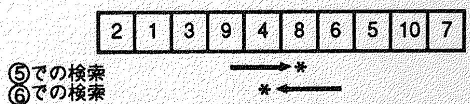


図12



図13　クイックソートの実際のコーディング

```
: void    qusort(initial_left , initial_right)
ソートすべき左端と右端の配列番号を受け取る．

: if(initial_right <= initial_left)    return;
ソート範囲が１以下であれば，何もしないで終了する．

: left  = initial_left;
: right = initial_right;
処理用ワークポインタの初期化

: center = base_data[left];
中央値はソートする範囲の配列先頭から取る

: do{
:     while(base_data[left] < center)        left++;
中央値より大きな値を配列の左側から探す

:     while(base_data[right] > center)    right--;
中央値より小さな値を配列の右側から探す

:     cmp = left - right;
:     if(cmp <= 0)
:     {
:         if(cmp != 0)
:             swap_data(left , right);
見つかった位置が交差していなければデータ交換を行う

:         left++;
:         right--;
次の検索に備えてポインタを更新しておく
:     }
: }while(left <= right);
今までの処理を両方のポインタが出会うまで行う

: if(initial_left < right)    qusort(initial_left , right);
※中央値によって分けられた左側の部分を今と同様に並べ替える．

: if(left < initial_right)    qusort(left , initial_right);
※中央値によって分けられた右側の部分を今と同様に並べ替える．
```

※の部分は，中央値によって振り分けられた２つのエリアの両側に対して同じことを繰り返していくために自分自身を再起的に呼び出す．これを配列の要素が１になるまで続ける．

リスト3　QSORT.MAK

```
.AUTODEPEND

#         *Translator Definitions*
CC = bcc +QSORT.CFG
TASM = TASM
TLINK = tlink

#         *Implicit Rules*
.c.obj:
  $(CC) -c {$< }

.cpp.obj:
  $(CC) -c {$< }

#         *List Macros*

EXE_dependencies =  ¥
  qsort.obj

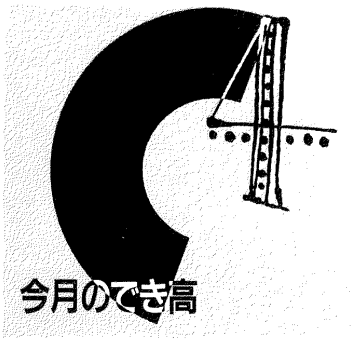
#         *Explicit Rules*
qsort.exe: qsort.cfg $(EXE_dependencies)
  $(TLINK) /x/n/c/P-/LA:¥BC¥LIB @&&|
c0s.obj+
qsort.obj
qsort
          # no map file
emu.lib+
maths.lib+
cs.lib
|

#         *Individual File Dependencies*
qsort.obj: qsort.c

#         *Compiler Configuration File*
qsort.cfg: qsort.mak
  copy &&|
-ff-
-vi
-IA:¥BC¥INCLUDE
-LA:¥BC¥LIB
-P-.C
| qsort.cfg
```

ほかにもブロック長が3くらいになった時点で残りのブロックをバブルソートのアルゴリズムで処理したほうが効率がよくなる場合もある．

このように，途中までクイックソートを使ってソートしていき，ブロック長がある程度の大きさまで収束してきたら別のアルゴリズムを使うという手法も考えられる．



今月はソート篇のまとめとして，最後に紹介した「クイックソート」を使ってWCOUNT.EXEの高速化をはかってみる．ソートのアルゴリズムを入れ替えるだけなので，先月号で作ったWCSUB.Cに含まれる「sort_data」という関数を改造するだけである．アルゴリズムの入れ替えとともに，ソートを進めていって，ブロック長が「2」になったときは，単純に2つのブロックを比較して必要なら入れ替えるという処理に切り替えて，ソートをより高速になるようにしてみる．リスト5が，改造後のWCSUB.Cである．

今月は，関数の内容を入れ替えるだけの改造なので，実行ファイルの作成や起動方法は先月とまったく同じである．メインソースの「WCOUNT.C」およびMAKEファイルも先月のものをそのままお使い願いたい．

表1に改造版のsort_data関数を解説しておく．

リスト4　QSORT.C

```
/* ================================================================

    QSORT.C         Copyright 1991 by JUSTECH-LABO. (H.kanoshima)

================================================================ */
#define DSIZE   10
#define DLEFT   0
#define DRIGHT  (DLEFT + DSIZE - 1)

int     base_data[DSIZE] = { 5, 1 , 3 , 9 , 4 , 8 , 6 , 2 , 10 , 7};

/* ================================================================ */

void    dump_data(char *guide)
{
int     i;

    printf("¥n%s",guide);
    for(i = DLEFT ; i < DSIZE ; i++)
        printf("%4d",base_data[i]);
    puts("");
}

/* ================================================================ */

void    swap_data(int x , int y)
{
int     tmp;

    tmp             = base_data[x];
    base_data[x]    = base_data[y];
    base_data[y] = tmp;
}

/* ================================================================ */

void    qusort(initial_left , initial_right)
{
int     center , cmp;
int     left , right;

    if(initial_right <= initial_left)   return;
    left  = initial_left;
    right = initial_right;
    center = base_data[left];

    do{
        while(base_data[left] < center)     left++;
        while(base_data[right] > center)    right--;

        cmp = left - right;
        if(cmp <= 0)
        {
            if(cmp != 0)
            {
                swap_data(left , right);
                dump_data("( - swap) ");
            }
            left++;
            right--;
        }
    }while(left <= right);

    if(initial_left < right)    qusort(initial_left , right);
    if(left < initial_right)    qusort(left , initial_right);

}

/* ================================================================ */

void    main(void)
{
    qusort(DLEFT , DRIGHT);
}
```

図14　QSORT.EXE実行画面

```
A>QSORT
( - swap)   2   1   3   9   4   8   6   5  10   7

( - swap)   2   1   3   4   9   8   6   5  10   7

( - swap)   1   2   3   4   9   8   6   5  10   7

( - swap)   1   2   3   4   7   8   6   5  10   9

( - swap)   1   2   3   4   5   8   6   7  10   9

( - swap)   1   2   3   4   5   6   8   7  10   9

( - swap)   1   2   3   4   5   6   7   8  10   9

( - swap)   1   2   3   4   5   6   7   8   9  10
```

リスト5　WCSUB.C

```
/* ==============================================================

    WSUB.C          Copyright 1991 by JUSTECH-LABO. (H.kanoshima)

============================================================== */
#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>

#define     WORDMAX     1000

typedef     enum    {BFALSE,BTRUE}  BOOL;

struct  _wlist{
    int     wcount;
    char    *wbody;
}wlist[WORDMAX];

struct  _wlist  *wlptr = wlist;

/* ==============================================================


============================================================== */
static  BOOL    wexist(char *sptr)
{
struct  _wlist  *wlp;

    for(wlp = wlist ; wlp < wlptr ; wlp++)
    {
        if(strcmp(wlp->wbody , sptr) == 0)
        {
            wlp->wcount++;
            return  BTRUE;
        }
    }
    return  BFALSE;
}

/* ==============================================================


============================================================== */
void    wset(char *sptr)
{
char    *cptr;

    if(wexist(sptr) == BTRUE)
    {
        putchar('!');
        return;
    }
    if(wlptr >= &wlist[WORDMAX])
    {
        printf("単語数が %d 個を超えました。¥n",WORDMAX);
        exit(1);
    }
    cptr = (char *)malloc(strlen(sptr) + 1);
    if(cptr == NULL)
    {
        printf("メモリが足りないので単語リストが作れません¥n");
        exit(1);
    }
    wlptr->wbody = strcpy(cptr , sptr);
    wlptr->wcount = 1;
    wlptr++;
    putchar('.');
}
/* ==============================================================


============================================================== */
void    swap_data(void *x , void *y , int dwidth)
{
char    *cptrx = x;
char    *cptry = y;

    while(dwidth--)
    {
    char    tmp;

        tmp = *cptrx;
        *cptrx++ = *cptry;
        *cptry++ = tmp;
    }
}
/* ==============================================================


============================================================== */
int     compar(const void *x , const void *y)
{
const struct _wlist *wptrx = x;
const struct _wlist *wptry = y;

    return  strcmp(wptrx->wbody , wptry->wbody);
}
/* ==============================================================


============================================================== */
int     (*q_compar)(const void * , const void *);
int     q_dwidth;

void    _qusort(char *initial_leftp , char *initial_rightp)
{
char    *centerp , *leftp , *rightp;
int     cmp;

    if(initial_rightp <= initial_leftp + q_dwidth)
    {
        if(initial_rightp == initial_leftp) return;

        if((*q_compar)(initial_leftp  , initial_rightp) > 0)
            swap_data(initial_leftp , initial_rightp , q_dwidth);
        return;
    }

    leftp  = initial_leftp;
    rightp = initial_rightp;
    centerp = leftp;

    do{
        while((*q_compar)(leftp  , centerp) < 0)    leftp  += q_dwidth;
        while((*q_compar)(rightp , centerp) > 0)    rightp -= q_dwidth;

        cmp = leftp - rightp;
        if(cmp <= 0)
        {
            if(cmp != 0)
            {
                swap_data(leftp , rightp , q_dwidth);
            }
            leftp  += q_dwidth;
            rightp -= q_dwidth;
        }
    }while(leftp <= rightp);

    if(initial_leftp < rightp)
        _qusort(initial_leftp , rightp);
    if(leftp < initial_rightp)
        _qusort(leftp , initial_rightp);
}

/* ==============================================================


============================================================== */
void    sort_data(void)
{
char    *i_left = (char *)wlist;

    q_compar = compar;
    q_dwidth = sizeof(struct _wlist);

    _qusort(i_left , i_left + ((wlptr - wlist - 1) * q_dwidth));
}
/* ==============================================================


============================================================== */
void    dump_word(void)
{
struct  _wlist  *wlp;
    printf("¥n**** ソート中 ****¥n");
    sort_data();
    for(wlp = wlist ; wlp < wlptr ; wlp++)
        printf("(%04d) - '%s'¥n" , wlp->wcount , wlp->wbody);
}
/* ==============================================================


============================================================== */
char    *wget(char **sptrptr)
{
static  char    getw[2048];
char    *sptrs , *sptrd;
int     cplen;

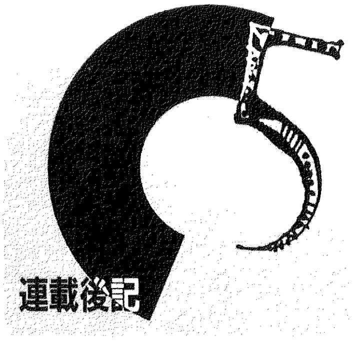
    sptrs = *sptrptr;

    sptrs += strspn(sptrs , " ,¥t¥n");
    if(*sptrs == '¥0')  return  NULL;

    sptrd = strpbrk(sptrs , " ,¥t¥n");
    if(sptrd == NULL)   cplen = strlen(sptrs);
    else                cplen = sptrd - sptrs;
    sptrd = getw;
    while(cplen--)
        *sptrd++ = *sptrs++;
    *sptrd = '¥0';
    *sptrptr = sptrs;

    return  getw;
}
```


ワンラインNEWS
松下電器産業と松下通信工業は共同で記憶容量が28MBの3.5インチFDD「JU-2521」を開発した。磁性材料に純鉄を採用，磁気ヘッドも改良し記録密度を向上させた。



今回の改造の効果を体感してみようと，25KB程の大きさのアセンブラソースファイルを新旧のWCOUNTで処理させてみた．同じファイルなのだが，先月号のWCOUNTだと，「＊＊＊＊ ソート中 ＊＊＊＊」が出てから5秒ほど止まったままになるが，今月のクイックソート版のほうはほとんど瞬時にリスト表示が始まる．驚くべき効果である．そして，その差はデータが大きくなるほど歴然としてくるであろう．ぜひ，ご自分の目でいろいろなテキストファイルを使って両者の違いを実際に確かめてほしい．

プログラムの高速化というと，ついコーディングの細かいところを直したりすることを考えがちだが，こうやってアルゴリズムの見直しをすることによって，大きな効果が得られる場合もあるということの典型的な例である．

さて次回からは，「検索」のアルゴリズムを紹介しながら，WCOUNTの単語リスト更新処理の際に，既存単語かどうかのチェックをしているところを見直していこうと思う．



表1

```
void    sort_data(void)
```

```
:       q_compar = compar;
```
クイックソートのための比較関数のポインタをグローバル変数に登録しておく．

```
:       q_dwidth = sizeof(struct _wlist);
```
単語リストの1件あたりのデータ長をグローバル変数に登録しておく．

```
:       _qusort(i_left , i_left + ((wlptr - wlist -1) * q_dwidth));
```
クイックソートの処理本体を呼び出す．与えるパラメータは，単語リストの先頭ポインタ（i_left）と最終ポインタである．

```
void    _qusort(char *initial_leftp , char *initial_rightp)
```

```
:       if(initial_rightp <= initial_leftp + q_dwidth)
```
呼び出されたときのブロック長が2以下になっていないかチェックする．

```
:           if(initial_rightp == initial_leftp)     return;
```
ブロック長が1であれば（左右のポインタが一致している状態）何もしないでリターンする．

```
:           if((*q_compar)(initial_leftp  , initial_rightp) > 0)
:               swap_data(initial_leftp , initial_rightp , q_dwidth);
:           return;
```
ブロック長が2であれば，ここで比較関数を呼び出して必要なら交換を終えてしまう．このあとのソート処理はする必要がない．

```
:       leftp  = initial_leftp;
:       rightp = initial_rightp;
```
処理用ワークポインタの初期化

```
:       centerp = leftp;
```
中央値は，ソートする範囲の先頭を使う．

```
:       do{
:           while((*q_compar)(leftp  , centerp) < 0)     leftp  += q_dwidth;
```
左側のポインタを配列の末尾に向かって増やしながら，中央値より大きな値が最初に見つかる位置を探す．

```
:           while((*q_compar)(rightp , centerp) > 0)     rightp -= q_dwidth;
```
右側のポインタを配列の先頭に向かって減らしながら，中央値より小さな値が最初に見つかる位置を探す．

```
:           cmp = leftp - rightp;
:           if(cmp <= 0)
:           {
:               if(cmp != 0)
:               {
:                   swap_data(leftp , rightp , q_dwidth);
```
右側と左側のポインタが交差していなければ，データの交換を行う．swap_data関数は先月と同じものが使用できる．

```
:               }
:               leftp  += q_dwidth;
:               rightp -= q_dwidth;
```
次の検索に備えて，ポインタを更新しておく．更新する値は単語リストの構造体1つ分のデータ長である．

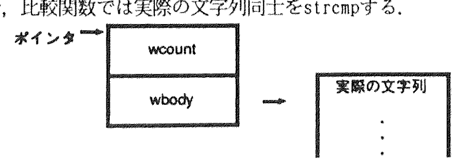
```
:           }
:       }while(leftp <= rightp);
```
今までの処理を，両方のポインタが出会うまで行う．

```
:       if(initial_leftp < rightp)
:           _qusort(initial_leftp , rightp);
```
中央値によって分けられた左側の部分にブロックができていたら，今と同様に並べ替える．

```
:       if(leftp < initial_rightp)
:           _qusort(leftp , initial_rightp);
```
中央値によって分けられた右側の部分にブロックができていたら，今と同様に並べ替える．QSORT.EXEのときに書いたように，この部分は再起呼び出しとなる．

```
int         compar(const void *x , const void *y)
```

```
:       return    strcmp(wptrx->wbody , wptry->wbody);
```
ポインタで与えられるのは，単語リストの先頭である．単語リストは下図のようになっているので，比較関数では実際の文字列同士をstrcmpする．



98活用アセンブラ技法

SCSIドライバ編
第2回

SCSIテープドライバの作成

星野 操

前回は，デバイスドライバのスケルトンを紹介し，それを利用したSCSIのインターフェースデバイスを作成した．今回は，それを利用したテープドライバを作成する．テープはランダムアクセス能力が低く，データの転送速度も低いなどの欠点があるが，一般に簡単な機構で大容量を実現し，かつビットあたりの単価が安いため，バックアップやデータ・プログラムの受け渡しなどによく使用されている．

前回のソースの修正

まず最初に前回の修正分です．dev-stdの280行目のあたり，リスト1の部分をリスト2のように変更してください．オリジナルがcondx.incを使用していなかったためcondx対応に書き直したのですが，その際に分岐形態を誤っていました．前のままだと，正常にオプションの解析ができなくなります．

SCSIデバイス

SCSIというと，まず思いつくのがディスクデバイスだと思います．読み書きのできるハードディスクはもちろんのこと，大容量のMOディスクや1回だけ書き込みのできる追記型光磁気ディスク，そして読み込み専用のCD-ROMなどのいろいろなディスクデバイスが外部バスにSCSIバスを採用し，接続されるようになっています．

これらのダイレクトアクセスデバイスは，ほとんど同一のコマンドセットをもっているので，基本的な部分，セクタの読み書き程度のレベルだと実際のメディアを意識することなく同一のコマンドを発行することが可能です．98でも，多少の制限はありますが，いろいろなメーカー・容量のドライブをバスに接続することが可能です．ただ，物理的・論理的には接続可能であってもボード上のBIOSの作りが悪いために接続できない場合も少なくありません．55ボードのIDチェックで他社のハードディスクが繋らないのは論外としても，MODESENSEコマンドの結果の解釈の仕方を手抜きしているために接続できないという場合があります．それでも，ハードウェア間の互換性はかなり高いといってかまわないと思います．

SCSIテープデバイス

これらディスクデバイス以外にも，いろいろなSCSIのデバイスが存在しています．たとえば，テープデバイスなどがそれにあたります．テープはランダムアクセス能力が低く，データの転送速度もあまり高くなく，信頼性もHDなどよりかなり劣るという欠点がありますが，一般に簡単な機構で大容量を実現し，かつビットあたりの単価がかなり安いので，常用ストレージではなく，バックアップやデータ・プログラムの受け渡しのためによく使用されます．

テープドライブの種類

テープドライブにも，いろいろな種類があります．まずメディアについていえば，WSやパーソナルコンピュータレベルで通常使われるのはDC600タイプのものとDC2000タイプのものがあります．ほかにTEACの，オーディオカセットと同一形状のデータカセットも存在しますが，これはいまひとつ不安定なような気がします．

大容量のほうに目を移すと，DATカセットや8mmビデオテープを利用したストリーマが存在し，それぞれ約1GB/2GBの容量をもっています．前者はフォーマットが規定されていないため，メーカー間でのメディアの互換性が薄いということですが，後者はほとんどEXABYTEの1社しか利用されていないため，メディアの流通性がよいということです．ですからWSなどで大容量のバックアップを必要とする場合には，EXABYTEを利用している場合が少なくありません．

それでも，普及度からいえばDC600シリーズにはかなわないでしょう．DC600は，主としてWSなどで使用されている新書版くらいのサイズのデータカセットです．UNIXシステムはフロッピーでインストールするのはほとんど不可能ですから，少なくとも1台のテープドライブは必須だといえます．もっとも最近は，インストールもCD-ROMドライブを使用できるようになってきたようですが．一方，DC2000シリーズはIBMコンパチなどの安価なテープストリーマに利用されているもので，サイズ的には名刺程度の大きさです．大きさには大きな違いがありますが，その構造は両者とも基本的にまったく同一です．

これらのデータカセットは特殊な，ベルトを利用した駆動系をもっています．これによって過大なテンションをかけることなくスムーズなテープ駆動を可能にしています．また構造上，テープの張り付きや巻き込み，たるみ，

リスト1

```
; check option key
cmp     al,'/'
@if     nz
        @break
@endif
```

リスト2

```
; check option key
cmp     al,'/'
@if     nz
        @continue
@endif
```

巻むらなどが発生しにくいようになっています．フィリップスタイプのデータカセットにくらべて，物理的な信頼性はかなり高いといってよいでしょう．また，テープ幅も1/4インチですから，フィリップスタイプよりは広くなっています．

テープデバイスのフォーマット

現在広く普及しているフォーマットは，DC600タイプではQIC-150，DC2000タイプではQIC-40またはQIC-80ということになります．その規格名から明らかなように，スタンダードなテープを使用するとQIC-150では150MB，QIC-40／80ではそれぞれ，40MB／80MBの記録が可能です．このほかにロングタイプのテープも存在し，それぞれ250MB／60MB／120MBの記録が可能となります．現在でも大容量化が指向されていて，DC600タイプで525MB記録可能なQIC-525のドライブ・テープもすでに入手可能になっています．最後には12GBまで記録可能にするということですが，はたしてそこまでいっても信頼性が保てるかどうかは疑問です．

テープの価格ですが，国内ではDC6150(QIC-150)で6000円程度，DC2120(QIC-80のロングタイプ)もほぼ同様な値段です．これがアメリカでは，ほぼ半分程度の値段となります．3.5インチタイプのMOメディアが非常に大胆な価格設定をしており，ビットあたりの容量では差が非常に小さくなってきています．

ドライブの値段にはまだ差がありますが，信頼性ではテープはMOにはかなわないでしょうから，QIC-150の競争力はだんだんと劣化することでしょう．テープドライブは，その記憶容量を大容量化(たとえば，QIC-525で525M／本）することによって競争力を維持していくことになるでしょう．QICタイプのテープは，下位規格のテープを読み取ることが容易に行えますから，大容量なものへのリプレースは比較的スムーズに移行すると思われます．

外部バスは，DC600タイプではSCSIを採用しているものと，QIC-02など専用外部バスを利用しているものとがあります．前者は一般にSCSIバスなら容易に接続可能であり，WSではこのタイプのものがほとんどです．一方，IBM PCコンパチなどに接続する場合には，専用カードを使用していることも少なくありません．これは，IBM PCには標準と呼べるSCSIカードが存在しておらず，特定のハードウェアを想定してドライバを添付することが困難であったからです．このため，一般には専用ボード＋専用ソフトでバックアップを可能にしているものが主流であり，WSとのデータ交換は困難となっています．

98の場合には，SCSIが標準化されていますから，ドライバが入手できればSCSIタイプのものが比較的容易に接続できます．たとえば，SUNなどのWSに接続されているものを，ちょっと外してきて接続する，などということも可能です(この場合，98のSCSIはシュリンクアンフェノールであり，SUNではSCSI-2が使用されているので，変換ケーブルが必要です)．

一方，DC2000タイプはFDの制御に使用する信号線をそのまま流用し，FDと同ような形式で操作できるようになっているものが多いようです．SCSIタイプのDC2000テープドライブというのは，聞いたことがありません．これは主としてIBM PCコンパチの，比較的低容量のハードディスクのバックアップに使用されるもので，安価にシステムを構成できるようにこのようなインターフェースを採用しているようです．ですから，これをほかのタイプのWSなどに接続することは非常に困難です．

バックアップに使用するソフトも専用のものを使うことがほとんどであり，流通性は非常に低くなってしまいます．もちろん，WSとのデータ交換に利用することはまったく期待できません．そのかわり，単に自分のハードディスクのバックアップ・リストアを実行するには非常に便利なようにできています．その使い勝手は，WSのテープなどまったく比較の対象にならないほどです．ですから，PCの世界で閉じてしまうことを覚悟するならば，DC2000タイプのドライブを採用することは十分に意味があります．とくに，システムを構成する値段が低減されるということには注意しなければいけません．

データの記録スピードは，後述する2150Sドライブで，QIC-150のときに大体6MB／minが最高速です．読み書きの方法によっては，容易にその半分以下に実効速度が落ちます．

DC2000タイプのドライブは，コロラドメモリシステムズのDJ-20に専用インターフェースボードを利用して，3MB／min程度が最高のようです．こちらはオンラインのデータ圧縮が可能なので，実効速度はそれより上がる場合もあります．専用インターフェースボードは，1Mbit／secの転送速度をもつため高速で読み書きできるのであり，通常のFDインターフェースを利用した場合には，その半分，500Kbit／secが最高となるので，実効速度もその半分程度となります．

まあいずれにしても，FDを用いてバックアップするよりも数段速く，かつディスク入れ替えなどの処理をしなくて済むという点で，現時点ではバックアップには最適のメディアといえるでしょう．

ヘッドポジショニングの柔軟性

9TRACKSなど大型のテープは，複数のヘッドをもち，複数のトラックを同時に読み書きします．しかし，QICタイプの記録方法はこれとはまったく異なっています．これらはただ1つのR/Wヘッドをもち，かつそれがステッピングモーターによって上下に移動できるようになっています．1つのトラックのデータを，テープの端から端まで駆動して読み書きしたあとは，次のトラックに物理的にヘッドを移動し，そしてまた端から端まで1度に読み書きするわけです．

この，ヘッドポジショニング機構のフレキシビリティによって，トラック間隔を比較的容易に変更することが可能です．線方向のデータ密度を上げるよりは，トラック密度を上げるほうがずっと容易です．実際，ハードディスクなどの高密度化も線方向の記録密度の向上よりは，トラック密度の向上のほうがはるかに寄与しているのです．

QIC-150の下にも容量の異なるいくつもの規格がありますが，これらの記録を読み込み(場合によっては書き込みも可)可能になっているのは，このフレキシブルなポジショニングによって適切なトラック位置に，ヘッドをもってくることが可能だからです．積層タイプのマルチヘッドだと，きっとこういう具合にはいかないでしょう．そのかわり，読み書きの速度はマルチヘッドよりは低下してしまいます．またQICタイプのテープにはシーク回数の制限がありますが，トラック密度をあまりに上げると1回の記録でテープが走行する回数が多くなり，バックアップに使用できる回数がかなり制限されてしまうということがおこりかねません．

さて，いずれにしろ98に接続し，バックアップメディアとして使用する場合にはDC600タイプのドライブ，外部バスはSCSIを使用しているものを利用することが必須となります．国内ではあまりこのようなテープドライブは見かけませんが，海外から直接入手することは可能です．また，前記のようにWSのテープドライブをちょっと借りてきて接続することもできないわけではありません．

テープ特有のSCSIコマンド

SCSIテープドライブでいちばんメジャーなのは，Archiveという会社の2150Sというタイプのものでしょう．SUNなどにもOEM供給されていると聞いています．このタイプのドライブで行える処理というのは，実は非常に限られています．INQUIREなどのSCSI一般で使用されるコマンドを除く，SCSIで利用可能なテープ特有のコマンドには次のようなものがあります．

【リワインド】

テープを巻戻し，ヘッドをデフォルトの位置に戻します．テープ終端は，そこに開けられている穴を光学的に検出することによって行います．テープを入れ替えたあとには，このリワインドを実行しておくほうがよいでしょう．

リワインドはデータのサーチを行いませんから，テープの走行は1パスで終了します．

【読み書き】

現在のヘッド位置から，指定ブロック数だけデータを読み書きします．1つのブロックを構成するバイト数は可変ですが，通常512バイトであるとして使用してもほとんど問題はないでしょう．読み書きが終了した後，ヘッドはその「後ろ」のポジションに移行します．

テープの終端では，走行方向を折り返し，かつヘッド位置を変更するという作業が物理的に必要になりますが，これらはすべてドライブの中で自動的に行われ，対応する処理はまったく必要ありません．だから，論理的にはテープは単一のトラックしかもっていないと考えることができます．逆にいえば，ヘッド位置を直接設定することにより，2次元のアクセスをすることが非常に困難であるということになります．これらの点が，ダイレクトアクセスデバイスと大きく異なる点です．

テープの読み書きをするためには，定速でテープを走行させないといけません．1度読み書きを中断すると，ヘッドは次のデータ位置よりも先に進んでしまいます．それを再起動するためには1度テープを巻戻し，それからウォームアップ走行を行い，安定してから読み書きを続ける，ということになります．

このバックトラック動作が入ると，テープの実効速度は極端に低下します．HDほどではないにしろ，テープの連続的な読み書き速度は決して遅くはありません．1sで100KBもの書き込みが可能です．ところが，バックトラックが行われると，1回に1s以上ものロスタイムが生じてしまいます．1ブロ

ック512バイトごとにこのバックトラック動作が起こった場合，どのようなことになるかわかるでしょう．

だから，テープを効率的に使用するためには，遅滞なくデータの引き取り・供給を行わないといけません．とくに98の場合，SCSIのバスの転送速度が遅いですから，これを行うのはなかなかむずかしいのです．

また，とくに書き込みにおいて注意しなければいけないのは，データのオーバーライトができないという点です．基本的には，テープには追記しかできません．データ記録位置の末端までシークしてから書き込む以外に手はないのです．もっとも，WORMデバイスとは異なり，テープの先頭から，つまりリワインドしたあとだとオーバーライトすることは可能です．ただ，この場合も前のデータとの共存を明確にはかることは困難です．だからテープデバイスでは，「ディレクトリ」を作成することが困難です．中身の一覧は，テープを最後まで走行させないと取得することができません．

ヘッドのダイレクトな位置ぎめ機構と，後述するファイルマークを利用すればより高度な管理ができる可能性はあります．しかし，WSクラスではそのような管理はまったく行っていません．IBMコンパチ用のDC2000シリーズのドライブ専用ソフトでは，これらディレクトリの管理を行うことが可能なので，実用性ははるかに高くなっています．

読み込みコマンドに関しては，逆方向の読み込みが指定できるようになっています．しかし，2150Sのインプリメントだと，1ブロックごとに正方向の読み込み・巻戻しを繰り返すので非常に遅く，実用には耐えません．

【ファイルマークの書き込み】

テープには，頭出しなどをするためにファイルマークというものを書き込むことができるようになっています．このファイルマークは，特殊な構造をしたデータブロックと考えても問題はないようです．たとえば，ファイルマークを超えて読み込みを実行しようとすると，そこでエラーが発生しますが，再び読み込みコマンドを発行するとファイルマーク以降のデータがそのまま読み出せます．

このファイルマークは，後述のSPACEコマンドで利用されます．

【ロード／アンロード】

メディアによっては，ヘッドの接触・非接触をコントロールするためなどに利用されるのでしょうが，QICタイプのテープではほとんど意味をもたないコマンドです．ただし，アーギュメントによって，リテンション指定を行うことができます．これはテープを端から端まで走行させ，それによってテープのたるみを取る，というものです．テープ走行をさせずに，出し入れをしたテープを使用する前に，このリテンションを実行するとよいといいます．

【消去】

テープの中身を消去します．2150Sでは，テープの部分消去はできず，全体消去のみ可能なようです．消去するためには，テープをリワインドしてからでないと実行できません．消去動作は1パスで終了することから，フルサイズの消去専用ヘッドが利用されているものと思われます．

【ヘッド位置の移動】

SPACEコマンドと呼ばれています．指定したブロックだけ前後にヘッドを移動したり，指定したファイルマークの数だけヘッドを移動したりできます．また，引数によって，データ記録領域の最後にテープを移動することも可能です．

正方向の移動は通常の読み込みと同様に高速ですが，逆方向の移動は結局逆方向の読み込みと同様で，非常に遅いオペレーションになります．たいていの場合，可能なら，巻戻してから正方向にスキャンするほうが速くなります．

ファイルマークを移動単位にする場合，ヘッドはファイルマークの直後に位置します．逆方向の検索においても，見つかったファイルマークの直前ではなく直後にヘッドは位置します．ですから，複数のボリュームを1つのテープに記録する場合，そのくぎり位置にファイルマークを打っておけば，そこまでテープを移動させることができます．ただし，ボリュームの内容の検索などを行った後，再びボリュームの先頭位置にヘッドを動かすために逆方向のファイルマークサーチを行うのは賢明ではありません．ものすごく時間がかかることになります．

ファイルマークの構造上，これを検索するためには通常のデータ読み取りと同様な処理をする必要があります．ですから，ファイルマークの検索をしても，検索スピードが上がるわけではありません．ビデオテープのVISSのように，高速でサーチができるわけではないのです．バックトラックさせないだけのデータ引き取りスピードがあれば，ファイルマークをサーチするのも，データを読み込むのも処理速度は変わりません．違うのは，データに対するサービスが必要であるかどうかという点だけです．

ワンラインNEWS
エヌ・エム・ビーセミコンダクターは92年半ばに16MビットDRAMのサンプル出荷を開始する．アクセス時間は40nsを予定しており，高速性で売り込む．

UNIXなどのシステムの場合，SCSIバスが開いていれば，このバスをほかのプロセスが使用できますし，CPUも別な処理に割り当てることができます．これに対して，DOSなどでは目的の処理が終わるまでは処理はサスペンドされますから，実際問題としてファイルマークサーチをしようが，データを読み捨てようが変わりはありません．

以上のことからわかるように，大きなボリュームのあとに，別なボリュームを追加すると，そのボリュームのアクセスには非常に時間と手間がかかることになります．ですからWSなどでデータをセーブする場合には，あまりがあっても1テープに1ボリュームしか入れない場合が少なくありません．

なお，同じサーチであっても，記録部位の最後端のサーチは非常に高速で，テープの走行は1パスですみます．これは，終端部を示す特別なマーカーが存在している(おそらくは，そのために1トラック使用されている)ために可能となっていると思われます．ですから，追記すること自体は，大した負荷にはなりません．それを必要が生じてアクセスするときに，たいへんになるわけです．

つまりシーケンシャルな読み込み，追記的な書き込みだけが基本的にサポートされており，それ以外のサーチなどはかなり制限が大きいということなります．

ですから，テープをDOSのデバイスとして使用する場合には，ブロックデバイスとしてランダムアクセスすることはほとんど不可能です．とくに既存の部位にオーバーライトできない，という点が致命的です．ですから，キャラクタデバイスとしてインプリメントすることになります．この場合，各種のテープコントロールはIOCTLを利用して行うのが適切でしょう．

TAPEデバイスドライバの作成

以上のような方針で作成されたTAPEデバイスドライバを，リスト3に示します．

このドライバは，やはりスケルトンDEVSTDと共用するものです．TAPE_DEVというシンボルを定義することによって，このファイルがインクルードされてテープ用のドライバが作成されるようになっています．

出力メッセージ

最初にあるのは起動時に出力されるメッセージです．次にあるmy_idはテープドライブのSCSI-IDであり，起動

リスト3

```
scsi_mes        db      'Scsi TAPE Driver',0dh,0ah,'$'
my_id   db      0
auto_rewind     db      0
error_command:
media_check:
build_bpb:
nd_read:
read_status:
read_flush:
write_status:
write_flush:
ioctl_in:
removable_media:
output_until_busy:
deinstall:
        mov     ax,e_bad_command
        jmp     interrupt_end
;
rewind_command  db      1,0,0,0,0,0
erase_command   db      19h,1,0,0,0,0
load_command    db      1bh,0,0,0,1,0
unload_command  db      1bh,0,0,0,0,0
retention_command       db      1bh,0,0,0,2,0
mark_command    db      10h,0,0,0,0,0
space_command   db      11h,1,0,0,0,0
space_b_command db      11h,0,0,0,0,0
ioctl_out:
        les     bx,packet
        les     bx,es:[bx].trans
        mov     al,es:[bx]
        mov     dx,es:[bx+1]
        cmp     al,'r'
        @if     z
                mov     si,offset rewind_command
        @else
        cmp     al,'e'
        @e_if   z
                mov     si,offset erase_command
        @else
        cmp     al,'l'
        @e_if   z
                mov     si,offset load_command
        @else
        cmp     al,'u'
        @e_if   z
                mov     si,offset unload_command
        @else
        cmp     al,'t'
        @e_if   z
                mov     si,offset retention_command
        @else
        cmp     al,'m'
        @e_if   z
                mov     si,offset mark_command
                mov     [si+4],dl
        @else
        cmp     al,'s'
        @e_if   z
                mov     si,offset space_command
                mov     [si+2],dh
                mov     [si+3],dh
                mov     [si+4],dl
                mov     byte ptr [si+1],1
                or      dl,dl
                @if     z
                        mov     byte ptr [si+1],3
                @endif
        @else
        cmp     al,'a'
        @e_if   z
                mov     auto_rewind,1
                jmp     ioctl_out_ok
        @else
        cmp     al,'n'
        @e_if   z
                mov     auto_rewind,0
                jmp     ioctl_out_ok
        @else
        cmp     al,'b'
        @e_if   z
                mov     si,offset space_b_command
                mov     [si+2],dh
                mov     [si+3],dh
                mov     [si+4],dl
        @else
                mov     ax,e_bad_command
                jmp     interrupt_end
        @endif
        mov     al,my_id
        mov     ah,0
        mov     cx,0
        les     di,sense_buffer_address
```

```

        call    scsi_prime_entry

        les     bx,packet
        les     bx,es:[bx].trans
        mov     es:[bx],ax
        mov     error_code,e_error
        or      al,al
        @if     nz
                jmp     ioctl_out_error
        @endif
        jmp     ioctl_out_ok

; r/w main part
inbuf   db      512 dup (?)
outbuf  db      512 dup (?)
in_len  dw      0
out_rest        dw      0

error_code      dw      ?
command_type    db      ?

ioctl_out_error:
read_write_error:
        les     bx,packet
        mov     es:[bx].count,cx

        cmp     al,2
        @if     z
                mov     ax,error_code
        @else
        and     ah,0fh
        cmp     ah,2
        @e_if   z
                mov     ax,e_not_ready
        @else
        cmp     ah,7
        @e_if   z
                mov     ax,e_write_protect
        @else
        cmp     ah,8
        @e_if   z
                mov     ax,e_no_sector
        @else
                mov     ax,error_code
        @endif
        jmp     interrupt_end

ioctl_out_ok:
        les     bx,packet
        les     bx,es:[bx].trans
        mov     word ptr es:[bx],0
read_write_ok:
open_close_ok:
        mov     ax,100h
        jmp     interrupt_end

read_write_block        proc
        mov     al,my_id
        mov     ah,command_type
        mov     bx,0
        mov     dx,1
        call    read_write_entry
        ret
read_write_block        endp

read:
        les     bx,packet
        mov     cx,es:[bx].count
        les     di,es:[bx].trans
        mov     error_code,e_read_error
        mov     command_type,0

        ; first check data
        mov     ax,in_len
        mov     si,offset inbuf+512
        sub     si,ax
        or      ax,ax
        @if     nz
                ; data exist in buffer
                cmp     cx,ax
                @if     be
                        ; read size < data buffer size
                        sub     in_len,cx
                        rep     movsb
                        jmp     read_write_ok
                @endif

                xchg    ax,cx
                sub     ax,cx
                rep     movsb
                xchg    ax,cx
        @endif

        ; cx:length, es/di:dist
        test    cx,0fe00h
        @if     nz
                ; block read
                push    cx
                mov     cl,ch
                mov     ch,0
                shr     cx,1
                call    read_write_block
                or      cx,cx
                pop     cx
                @if     z
                        jmp     read_write_error
                @endif
        @endif

        and     cx,1ffh
        @if     nz
                ; odd size
                push    cx
                push    es
                push    di
                mov     cx,1
                push    cs
                pop     es
                mov     di,offset inbuf
                call    read_write_block
                or      cx,cx
                pop     di
                pop     es
                pop     cx

                @if     z
                        jmp     near ptr read_write_error
                @endif

                mov     in_len,512

                sub     in_len,cx
                mov     si,offset inbuf
                rep     movsb
        @endif
        jmp     read_write_ok

;-----------------------------------
write_copy      proc
        xchg    si,di
        push    es
        push    ds
        pop     es
        pop     ds
        rep     movsb
        push    es
        push    ds
        pop     es
        pop     ds
        xchg    si,di
        ret
write_copy      endp

write:
write_verify:
        les     bx,packet
        mov     cx,es:[bx].count
        les     di,es:[bx].trans
        mov     error_code,e_write_error
        mov     command_type,1

        ; check data
        mov     ax,out_rest
        mov     si,offset outbuf+512
        sub     si,ax
        or      ax,ax
        @if     nz
                ; data exist in buffer
                cmp     cx,ax
                @if     b
                        ; write size < data buffer size
                        sub     out_rest,cx
                        call    write_copy
                        jmp     read_write_ok
                @endif

                xchg    ax,cx
                sub     ax,cx
                call    write_copy
                xchg    ax,cx

                push    cx
                push    di
                push    es
                mov     cx,1
                mov     di,offset outbuf
                push    cs
                pop     es
                call    read_write_block
                pop     es
                pop     di
                pop     cx
                mov     out_rest,0
        @endif

        test    cx,0fe00h
        @if     nz
                ; block write
                push    cx
                mov     cl,ch
                mov     ch,0
                shr     cx,1
                call    read_write_block
                or      cx,cx
                pop     cx
                @if     z
                        jmp     read_write_error
                @endif
        @endif

        and     cx,1ffh
        @if     nz
                mov     si,offset outbuf
                mov     out_rest,512
                sub     out_rest,cx
                call    write_copy
        @endif
        jmp     read_write_ok

```

時に設定されます．これは，LUNとマルチプレクスされている値です．auto_rewindはテープデバイスがクローズされたときに自動的にリワインドを実行するかどうかを決めるフラグで，IOCTLで設定されます．それから，サポートされていないブロックデバイス関係などのファンクションエントリがあり，それらはすべてエラーで返るようになっています．

SCSIコマンドテーブル

次に，サポートされているSCSIコマンドのテーブルがあります．読み書きについては，SCSIデバイスのそれをそのまま流用するのでここでは記述されていません．ここで定義されているのは，そのほかのコントロールコマンドです．コマンドの内容については前記したのでわかると思います．ここで，SPACE･MARK関係のコマンドだけは，その内容が変更されることがあります．とくに，テープエンドまでのスキップもSPACEであるということに注意します．

IOCTLについては，IOCTL_OUTのみサポートされており，これを用いてテープコントロールが可能となっています．IOCTLの引数は，1文字のコマンドキャラクタと1ワードの値からなっています．リワインドやイレーズの場合は，SIに指定のコマンド列をセットするだけですが，MARKの場合には書き込まれるファイルマークの個数をコマンドパケットに設定します．

ファイルマーク単位のスペースの場合も同様ですが，引数が0である場合にはパケットのコマンドタイプの部分を書き換え，テープエンドまでのシークコマンドに変更します．ブロック単位の場合もほぼ同様な処理をします．オートリワインドの設定については，コマンドによってそれぞれ0か1を変数に代入します．これが非ゼロだと，オートリワインドが有効になります．

コマンドパケットがSIに設定されると，あとはSCSI-IDなどのパラメータをレジスタにセットし，SCSIのエントリを呼び出します．このエントリは，DEVSTDの中で定義されていて，起

```
;--------------------------------

open_count      dw      0
old_open_count  dw      ?
old_21_vector   dd      ?

device_open:
        inc     open_count
        jmp     open_close_ok

device_close:
        ; abandone data in buffer
        mov     in_len,0
        cmp     out_rest,0
        @if     nz
                ; output flush
                mov     cx,1
                mov     di,offset outbuf
                push    cs
                pop     es
                call    read_write_block
                mov     out_rest,0
        @endif

        ; rewind check
        @mc
                dec     open_count
                @cond   z
                cmp     auto_rewind,0
                @cond   nz
        @if
                mov     si,offset rewind_command
                mov     al,my_id
                mov     ah,0
                mov     cx,0
                les     di,sense_buffer_address

                call    scsi_prime_entry
        @endif
        jmp     open_close_ok

old_21:
        jmp     cs:old_21_vector

new_21:
        sti
        cmp     ah,3dh
        jz      new_21_2
        cmp     ah,3ch
        jnz     old_21
        new_21_2:
        push    cs:open_count
        pop     cs:old_open_count

        pushf
        call    cs:old_21_vector

        pushf
        push    bx
        mov     bx,cs:open_count
        cmp     bx,cs:old_open_count
        @if     nz
                push    ax
                push    dx
                mov     bx,ax
                mov     ax,4400h
                pushf
                call    cs:old_21_vector
                mov     dh,0
                or      dl,20h
                mov     ax,4401h
                pushf
                call    cs:old_21_vector
                pop     dx
                pop     ax
        @endif
        pop     bx
        popf
        retf    2

end_label:

ex_init proc    near
        ; first check options
        push    cs
        pop     es
        mov     bx,offset options
        @begin
                ; get argument value
                push    bx
                push    es
                les     bx,[bx].o_arg
                call    atoh
                pop     es
                pop     bx

                mov     dl,[bx].o_char
                cmp     dl,'i'
                @if     z
                        mov     [bx].o_char,0
                        or      my_id,al
                @else
                cmp     dl,'l'
                @e_if   z
                        mov     [bx].o_char,0
                        mov     cl,5
                        shl     al,cl
                        or      my_id,al
                @endif

                add     bx,5
                cmp     bx,offset options_tail
        @until  nc

        mov     ax,3521h
        int     21h
        mov     word ptr old_21_vector,bx
        mov     word ptr old_21_vector+2,es

        mov     ax,2521h
        mov     dx,offset new_21
        int     21h

        ret
ex_init endp

```

動時に自動的に値が設定されるDWORDの変数です．これを経由して，SCSIデバイスのエントリを呼び出します．

それから，エラーチェックをして復帰します．変数error_codeは，SCSIから直接返されたエラー以外のエラーに基づいて呼び出し元に返すエラーコードを設定します．

読み書き用バッファ

その後ろには，読み書き用のバッファがあります．テープはブロッキングされているので，バイト単位で読み書きはできませんが，キャラクタデバイスとして使用するのでデバイスドライバの呼び出しの場合にはバイト単位での読み書きが要求されることがあります．これらを処理するために，1ブロックサイズの読み書きバッファが必要になるのです．同時に，バッファ内のデータ長を示す変数も必要になります．

エラー処理ルーチン

次にあるのはエラー処理ルーチンで，SCSIのセンスキーの値に基づいてDOSのエラーコードを生成します．書き込みがされていないテープを読み込むと，エラーコード8のBLANK CHECKが発生するので，この場合にはSECTOR NOT FOUNDタイプのエラーを返すことにします．

該当するエラーがなければ，変数error_codeに格納されている値をエラーコードとして採用します．

read_write_block

次のread_write_blockは，レジスタにデフォルトの値を設定して読み書き用のエントリを呼び出すものです．command_typeには読み書きの方向が設定されています．DXに1を設定しているのは，ブロック単位でのアクセスを示すためののものです．

readエントリは，コマンドパケットから各種のデータを取得し，さらにワークエリアの初期化をします．それから，現在はんぱなデータがバッファに残っているかどうかをチェックし，それがあれば最初に転送をします．その場合には，要求転送長も同時に処理します．次に，ブロック単位の直接転送を行います．バイト長からブロック長への変換をし，それからread_write_blockを呼び出します．読み込みデータ長が0であれば，エラー終了します．

最後に，ブロック未満の部分の転送要求があれば，1ブロックだけ読み込みを行い，それからバッファ経由でデータを転送します．この場合，あまりの部分がバッファに残ることになります．

書き込みの場合は，その逆になりますが，レジスタの関係からブロック転送用にwrite_copyという手続きを1つ作っています．writeの中では，同様に現在はんぱがあれば，まずそれをブロックに形成し1ブロック書き出そうとします．次に，ブロック単位で書き出しを行い，最後にはんぱな部分が残っていれば，これをバッファに転送します．

デバイスオープン/クローズ処理

その後は，デバイスオープンなどの処理です．テープに複数のボリュームを格納している場合には，1つのボリュームにアクセスしたあとヘッドの位置はそこに残っていてもらわないと困ることになります．これに対して，1つのテープに1つのボリュームしか格納しない場合には，アクセスが終わったあと自動的に巻戻しをしてくれたほうが便利です．これをIOCTLで設定できるようにしてあるのですが，そのためにはデバイスクローズをトラップし，テープがクローズされたということを知らないといけません．

また，テープをバックアップに使用する場合，バイナリーコードに対してトランスペアレントでないと困ります．しかし，キャラクタデバイスはオープンした直後は必ずcoockedモードになってしまいます．このため，int21のDOSエントリをトラップし，テープのオープン直後にIOCTLでrawモードへのモード変更を行わないといけません．このためにも，デバイスオープン処理を把握する必要があります．ですから，ここでデバイスオープン・クローズをサポートしていますが，それはリムーバブルディスクにおいての不正なディスク交換検出目的とは異なった使い方をしている，ということになります．

まず，デバイスオープンはオープンカウントをインクリメントするだけの処理です．このカウンターのチェックはINT21の呼び出しルーチンで実行されます．次に，デバイスクローズはこのオープンカウントをデクリメントします．そして，読み書きバッファの内容をフラッシュアウトします．それから完全にクローズされた場合，かつauto_rewindが非ゼロであればリワインドコマンドを自動的に実行しテープを巻き戻します．

前記のようにrawモードへの設定を行うため，int21もトラップしています．そしてオープン関係のファンクションでなければ直接元のint21を実行

します．そうでなければ，現在のopen_countをセーブし，int21を呼び出します．

そして，open_countの値に変更があったかどうかをチェックし，変更があった場合には自動的にIOCTLのコマンドを発行します．そして復帰するわけですが，フラグレジスタに関しては簡易的な設定をしています．

初期化ルーチン

ここまでが常駐する必要のある部分であり，そのあとに切り分け部分を示すラベルと初期化のルーチンが用意されます．初期化のルーチンはｉオプションとｌオプションを認識し，これらがそれぞれSCSI-IDとLUNを指定できるようになっています．デフォルトは，おのおの０です．たとえば，

```
device = xtape.sys /l3
```

と指定すれば，テープドライブのSCSI-IDは３になります．オプションの解析が終了した後，前記のint21のベクタのすり替えを行い，初期化は終了します．

プログラムの作成

実際にテープデバイスを作成するには，

```
TASM /DTAPE_DEV DEV-STD,XTAPE
TLINK /T XTAPE,XTAPE.SYS
```

などと実行すればよいでしょう．このデバイスドライバをロードするに先立

リスト４　MT. C

```
#include <stdio.h>
#include <dos.h>

void usage()
{
    fprintf(stderr, "Usage: mt -[relutmsban]¥n");
    fprintf(stderr, "where,¥n");
    fprintf(stderr, "   (R)ewind tape¥n");
    fprintf(stderr, "   (E)rase tape¥n");
    fprintf(stderr, "   (L)oad tape¥n");
    fprintf(stderr, "   (U)nload tape¥n");
    fprintf(stderr, "   re(T)ension tape¥n");
    fprintf(stderr, "   write file (M)ark n¥n");
    fprintf(stderr, "   (S)pace tape n¥n");
    fprintf(stderr, "   space (B)lock tape n¥n");
    fprintf(stderr, "   (A)uto rewind tape¥n");
    fprintf(stderr, "   (N)ot auto rewind tape¥n");

    exit(1);
}

void main(int argc, char **argv)
{
    char iobuf[3];
    char *error_mes[] = {
        "Access Error",
        "Recovered Error",
        "Device Not Ready",
        "Medium Error",
        "Hardware Error",
        "Illegal Request",
        "Unit Attention",
        "Data Protect",
        "Blank Check"
    };
    int i;

    _DX = "STAPE";
    _AX = 0x3d00;
    geninterrupt(0x21);
    i = _AX;
    if (_FLAGS & 1) {
        fprintf(stderr, "Open error¥n");
        exit(1);
    }

    if (argc != 2 || argv[1][0] != '-')
        usage();

    switch (argv[1][1]) {
        case 'r':
        case 'e':
        case 'l':
        case 'u':
        case 't':
        case 'm':
        case 's':
        case 'a':
        case 'n':
        case 'b':
            iobuf[0] = argv[1][1];
            iobuf[1] = atoi(argv[1] + 2);
            iobuf[2] = atoi(argv[1] + 2) >> 8;
            break;
        default:
            usage();
    }

    _DX = iobuf;
    _CX = 3;
    _BX = i;
    _AX = 0x4403;
    geninterrupt(0x21);

    _AX = 0x3e00;
    _BX = i;
    geninterrupt(0x21);

    if (iobuf[0] == 2) {
        if (iobuf[1] < sizeof(error_mes) / sizeof(int))
            fprintf(stderr, "MT:%s¥n", error_mes[iobuf[1]]);
        else
            fprintf(stderr, "MT:Access Error¥n");

        exit(1);
    }
    fprintf(stderr, "done!¥n");
    exit(0);
}
```

ワンラインNEWS
横河・ヒューレット・パッカードは記憶容量が2.35GBの5.25インチHDD「HP C3010」を開発，販売を開始．日立製作所の3.7GBに次ぐ大容量を実現している．

って，必ずXSCSI.SYSをロードしておく必要があります．

テープコントロールプログラム

IOCTLを利用して，テープをコントロールするプログラムも載せておきましょう．MT.Cがそれです(リスト4)．

TCのレジスタ疑似変数を利用して，簡潔にまとめてしまっています．サポートしている機能は最初のusageで明らかでしょうが，要するにXTAPEのIOCTLがサポートしている範囲ということになります．このうち，m／s／bのコマンドの場合は10進数値が引数として与えられることになります．たとえば

```
MT -m2
```

とすると，ファイルマークが2個書き込まれます．sコマンドの引数が0であると，テープエンドまでのシークであるとみなされます．

main最初のerror_mesは，SCSIのセンスキーに対応するエラーメッセージです．mainは最初にデバイスSTAPEをオープンし，それからオプションを解析します．有効なオプションであれば，ワークエリアにオプション文字と引数を格納し，IOCTLを実行します．それからデバイスをクローズします．IOCTL実行時にDOSのエラーコードは理解されません．ですから，XTAPEの中で，引数バッファにSCSIのセンスキーの値を返すようにしてあります．これらをチェックし，異常であればメッセージを出力します．

バックアップの実際

実際にこれらを使用して，ディスクのバックアップを取る場合には，TARコマンドを使用するのがいちばん簡単です．TARは，簡単なヘッダを付けるだけでファイルの中身を生のままダンプしていくソフトと考えてよいでしょう．その構造のシンプルさのため，WSではデータ交換にひんぱんに使用されますが，バックアップ目的で使用することも可能です．これを利用すれば，WSとのデータ交換も行えます．

DOSで動くTARというと，GNUのTARやmixなどのネットワークにいらっしゃるKmoriさんの作成されたKMtarなどがあります．GNUTARは，漢字ファイル名の処理に難があります．またファイルの数が非常に多いと暴走状態に入ることがあるようです．KMtarの新しいものは，そのあたりはソリッドに動くようです．古いものは，バッファリングの関係から実用的な速度では稼動しません．

TARを利用してBドライブのバックアップをする場合には，

```
tar cvf stape b:*.*
```

と起動すればOKです．もちろん，その前にMTコマンドでリワインドしておきます．fオプションでセーブ先のデバイス名が指定できるので，ここではテープデバイスの名前であるstapeを指定すればよいわけです．以前s$tapeという名前を使ったことがあったのですが，こうするとエラーが発生した場合にDOSのメッセージが'$'キャラクタで中断されてしまうことになります．エラーの発生する可能性のあるデバイスの場合は，名前に'$'を含ませてはいけません．

さらに追記する場合には，MTコマンドでファイルマークを打ち，tar cvfでセーブを繰り返せばよいでしょう．リストを取る場合には，tar tvfであり，内容を復元する場合にはtar xvfです．特定のボリュームまでスキップする場合には，MTのsオプションで可能ですが，スピードの面だけからいうとtar tvfで前のボリュームの内容をスキャンしても，あまり変わりありません．

これらは，いずれもバックトラックなしでバックアップを行えるほど高速には稼動しません．数秒に1回バックトラックが入るという感じですが，実効速度はそれだけで最大値の半分，3MB／min程度に落ちてしまっています．単にバックアップをするだけならディスクのイメージをダンプするほうが簡単であり，高速です．次回には，そのようなソフトと，ブロックデバイスタイプのデバイスドライバの作成について触れてみようと思います．

情報処理技術者試験

第2種合格！トレーニング

第11回　問題と解答 ―― 午前の部

静岡情報ビジネス専門学校　望月照夫

今月から２回にわたり，試験前の対策として，午前と午後の部にわけ，おさえておきたい問題とその解答を掲載します．

問１

浮動小数点に関する次の記述中の［a］～［d］に入れるべき適切な答を解答群の中から選べ．

図１のような４バイトで表す浮動小数点数がある．

＊仮数部の数値は絶対値で表す．

＊指数部のｅは，指数が16のｎ乗をｎ＋64の値で表す．

＊数値全体の正負を第０ビットで表し，正ならば０，負ならば１を立てる．

したがって，この形式では浮動小数点数は，図２のように表現される．

いま65535をこの形式で表現すると，［a］となり，11/4096は［b］となる．この２つの数を加算する時，指数部は大きな数に合わされるため，［b］は［c］となる．

この時仮数部の有効桁数が落ちることがある．これを［d］と呼ぶ．

［a～cの解答群］

ア．C4FFFF00　イ．44FFFF00

ウ．3EB00000　エ．44000000

［dの解答群］

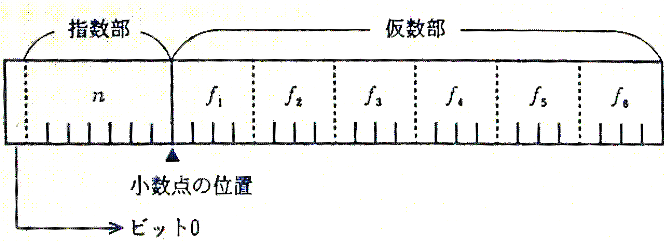
ア．打切り　イ．けた落ち

ウ．情報落ち

解答

ａ－イ　ｂ－ウ　ｃ－エ　ｄ－ウ

●問１－図１



●問１－図２

$$16^n \times (f_1/16 + f_2/16^2 + f_3/16^3 + f_4/16^4 + f_5/16^5 + f_6/16^6)$$

問２

主記憶およびアキュームレータの内容が図１のようになっているとき，ａ～ｅのアドレス指定方式で加算命令が実行された時のアキュームレータの値を解答群の中から選べ．

●問２－図１

95番地	100
96	101
97	102
98	103
99	104
100	105
101	106
102	107

アキュームレータ	5
ベースレジスタ	95
指標レジスタ１	1
指標レジスタ２	2

a．	命令	95	
b．	命令	100	
c．	命令	95	
d．	命令	3	
e．	命令	95	2

※命令形式は

命令	アドレス	指標レジスタ番号

とする．

a．即値アドレスの場合

b．直接アドレスの場合

c．間接アドレスの場合

d．相対アドレスの場合

e．指標アドレスの場合

[解答群]

ア．100　イ．101　ウ．102　エ．103

オ．104　カ．105　キ．106　ク．107

ケ．108　コ．109　サ．110

解答

a－ア　b－サ　c－サ　d－ケ

e－ク

問 3

次の文章と図の中のａ～ｈに入れるべき字句を解答群の中から選べ．

図１は，一般的な汎用コンピュータの記憶装置を示したものである．このように，高速，小容量の装置と低速，大容量の装置を組み合せて，見掛け上アクセス時間が[ａ]，[ｂ]の記憶装置を実現しようとするのが，[ｃ]の考え方である．

[a～cの解答群]

ア．仮想記憶　イ．短く　ウ．長く

エ．小容量　オ．大容量　カ．記憶階層

[d～iの解答群]

ア．数十 ms　イ．数百 ns　ウ．数百 ms

エ，オ．数 ns　カ．数百 KB

キ．数十 MB　ク．十～百 MB

ケ．数 MB

解答

a－イ　b－オ　c－カ　d－イ

e－ア　f－カ　g－ク　h－キ

●問 3 －図 1

記憶装置	アクセス時間	記憶容量
キャッシュメモリ	10～100ns	[ｆ]
主記憶	[ｄ]	[ｇ]
ディスクキャッシュ	100μs～10ms	[ｈ]
磁気ディスク	[ｅ]	数 GB／装置
磁気テープ	10～100s	数百 MB

問 4

キャッシュメモリに関する次の記述中，[ａ]～[ｃ]に入れるべき適当なものを，解答群の中から選べ．

CPU からのアクセス要求に対し，その情報がキャッシュメモリに存在する割合をヒット率と呼び，存在しない割合を NFP(Not Found Probability)と呼ぶと，両者の間には次の関係が成り立つ．

[ａ]

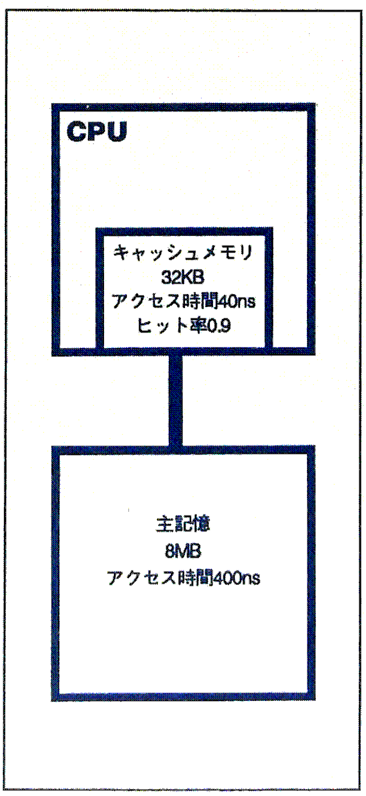
また，キャッシュメモリのアクセス時間を Tc，主記憶のアクセス時間を Tm とすると，

記憶装置の平均アクセス時間Ｔは

Ｔ＝［ｂ］と表される．

いま図１のようなコンピュータがあったとき，平均のアクセス時間は[ｃ]となる．

●問 4 －図 1



[a，bの解答群]

ア．NFP・Tm＋ヒット率・Tc

イ．ヒット率＝1/NFP

ウ．NFP・Tc＋ヒット率・Tm

エ．ヒット率＝1－NFP

[cの解答群]

ア．76ns　イ．364ns　ウ．44ns

解答

a－エ　b－ア　c－ア

問 5

次の記述を読み［ａ］～［ｅ］に適当なものを解答群の中から選べ．

磁気ディスク中のデータを１ブロックずつ読み出し，そのまま磁気テープに書き込む．ただしデータブロックの長さは，19200バイトで10,000件ある．磁気テープと磁気ディスクの仕様は図１に示すとおりである．チャネルは各装置に設置されており，並列処理を行

なう．また，CPUでの処理時間は無視するものとする．

磁気ディスク装置からの1ブロックのアクセス時間は［a］である．

磁気テープ装置への1ブロックのアクセスは時間は［b］である．

つまり，磁気ディスクへのアクセス時間が磁気テープへのアクセス時間と比べ［c］ので，全体の処理時間は，［d］のアクセス時間を件数倍したものとなり，［e］となる．

［a，bの解答群］

ア．16ms　イ．81ms　ウ．75ms

エ．56ms　オ．40ms

［c，dの解答群］

ア．長い　イ．等しい　ウ．短い

エ．磁気テープ装置

オ．磁気ディスク装置　カ．両方

［eの解答群］

ア．約13.5秒　イ．13.5分

ウ．1.35時間

解答

a－エ　b－イ　c－ウ　d－エ

e－イ

●問5－図1

磁気テープ装置	
記録密度	64列／mm
ブロック間隔	15mm
テープ走行速度	4m／秒
起動・停止時間	6ms

磁気ディスク装置	
シリンダ数	800シリンダ
トラック数／シリンダ	20トラック
トラック容量	24,000バイト
回転速度	3,000rpm
平均シーク時間	30ms

問6

リスト処理に関する次の記述のうち［a］～［i］に入るべき適当なものを解答群の中から選べ．

図1のような双方向リストでは，次ポインタには次のデータの要素番号が，前ポインタには，前のデータの要素番号が入っている．

このリストの先頭の要素番号は［a］であり，最後の要素番号は，［b］である．このリストの要素番号50に置かれている「小田原」を削除するには，［c］の次ポインタを［d］に，［e］の前ポインタを［f］に変えるとともに，「小田原」の削除ビットを1にする．また，要素番号130に置かれた「富士」を「三島」と「静岡」の間に挿入する場合は，「三島」の次ポインタを［g］，「静岡」の前ポインタを［g］，「富士」の前ポインタを［h］，次ポインタを［i］とする．

［a，b，d，f～iの解答群］

ア．10　イ．30　ウ．50　エ．70

オ．90　カ．110　キ．130

［c，eの解答群］

ア．東京　イ．新横浜　ウ．小田原

エ．熱海　オ．三島　カ．富士

キ．静岡

●問6－図1

要素番号	データ部	削除ビット	次ポインタ	前ポインタ
10	東京	0	110	0
30	熱海	0	70	50
50	小田原	0	30	110
70	三島	0	90	30
90	静岡	0	0	70
110	新横浜	0	50	10
130	富士	0	－	－

解答

a－ア　b－オ　c－イ　d－イ

e－エ　f－カ　g－キ　h－エ

i－オ

問7

オペレーティングシステムに関する［a］～［f］の機能に最も関連の深いものを解答群の中から選べ．

［a］オペレーティングシステムは，利用者が特定の入出力装置を意識することなくプログラムを作成できること，すなわちディバイス・インデペンデンスを保証する．

［b］オペレーティングシステムは，割込み処理プログラムの実行後，実行可能状態タスクの中から最高の優先度のものを選択し，実行権を与える．

［c］オペレーティングシステムは，ジョブの終了時にジョブの処理に要した時間，主記憶の占有時間，入出力装置の使用時間などをリスト出力する．

［d］オペレーティングシステムは，入力ストリームを読み取り，解読し，一連のジョブの実行をスケジュールしジョブの実行を起動させる．

［e］オペレーティングシステムは，タスクが実際に必要とするページのみを必要な時点で主記憶にロードし，アドレス空間が主記憶容量よりも大きなタスクでも実行を可能にしている．

［f］オペレーティングシステムは，タスクが複数資源の排他利用を行う場合には，その占有の順序をタスク同士で一致させる必要がある．この順序を誤ると，お互に相手の確保している資源を要求し合って永久に待ち状態になることがある．

［解答群］

ア．ダイナミックリンキング
イ．仮想記憶
ウ．ジョブスケジューラ
エ．マスタースケジューラ
オ．データ管理機能
カ．デッドロック　キ．課金情報
ク．ディスパッチャ　ケ．PSW

解答

a－オ　b－ク　c－キ　d－ウ
e－イ　f－カ

問8

計算機システムの信頼性に関する次の記述のうち，［a］～［g］に入れるべき適当なものを解答群の中から選べ．

システムの信頼性は，システムの時間的安定性を表し，MTBFで示される．さらに，MTBFは次の式で求められる．

MTBF＝［a］

また，保守性とは，故障発生時の保全の容易さをいい，通常［b］で表される．

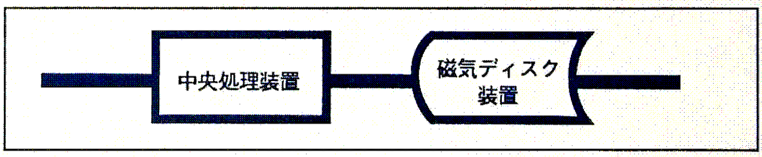
さらに可用性は，信頼性と保守性の総合的機能であり，全体の時間のうちシステムが正常に機能する時間を示し，一般的に次式で導かれる［c］により表現される．

［c］＝［d］

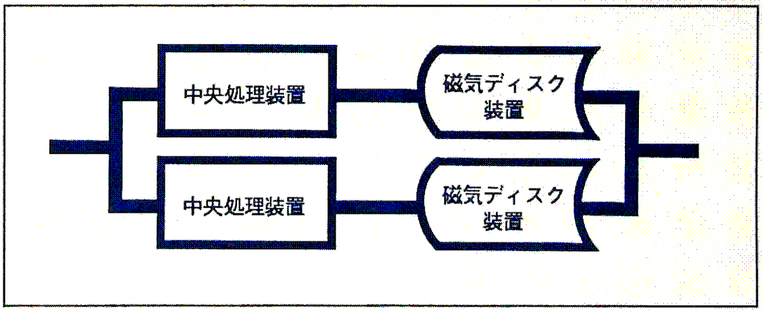
以上3つを総称して［e］と呼ばれる．

いま，図1のようなシステムがあった場合，このシステムの［c］は［f］となるが，図2のごとくシステム全体を二重化することにより，［c］は［g］となる．ただし中央処理装置，磁気ディスク装置とも，［c］＝0.9とする．

●問8－図1



●問8－図2



［a，dの解答群］

ア．MTBF／（MTBF＋MTTR）
イ．MTTR／（MTBF＋MTTR）
ウ．累計使用時間／故障件数
エ．累計使用時間×故障率

［b，c，eの解答群］

ア．故障率　イ．ヒット率
ウ．MTTR　エ．故障件数
オ．MTBF　カ．累計使用時間
キ．稼働率　ク．RAS　コ．CASE

［f，gの解答群］

ア．0.9　イ．0.81　ウ．0.999
エ．0.964

解答

a－ウ　b－ウ　c－キ　d－ア

e－ク　f－イ　g－エ

問9

中央処理装置の性能に関する次の記述中［a］～［d］に入れるべき適当なものを解答群の中から選べ．

計算機の性能の指標として［a］が用いられる．

これは，1秒間に何百万命令を実効できるかを示すもので，あらかじめその計算機に備っている命令セットのなかから，一部の命令を重み付けした割合で組合せたモデルにしたがって計算されることが多い．

いま図1に示すモデルにより［a］を求めると，約［b］となる．また，乗算と除算の実効速度がそれぞれ1/4に短縮されると［a］は約［c］となり，平均命令実行時間は約［d］％短縮される．

［解答群］

ア．MFLOPS　イ．MIPS　ウ．BPS
エ．1.0　オ．5.3　カ．0.53
キ．0.66　ク．23.6　ケ．19

●問9－図1

命令の種類	演算時間	相対頻度
ロード／ストア	1.0マイクロ秒	35%
加減算	1.5マイクロ秒	30%
比較／分岐	1.6マイクロ秒	25%
乗算	8.0マイクロ秒	3％
除算	12.0マイクロ秒	2％
その他	4.0マイクロ秒	5％

解答

a－イ　b－カ　c－キ　d－ケ

問10

データ通信に関する次の記述のうち［a］～［i］に適当なものを解答群の中から選べ．

基本形データ伝送制御手順は，10種類の伝送制御キャラクタを使用し，次に示す5つの段階を経てデータ伝送を行う．

フェーズ1　［a］
2　データリンクの確立
3　［b］
4　［c］
5　回線の切断

このうち，データリンクの確立の方法としては，［d］方式と［e］方式の2つがある．［d］方式は回線争奪方式とも呼ばれ，ポイント・ツー・ポイント接続で用いられ，双方の局が対等の立場で制御権を行使する．［e］方式は集中正義方式とも呼ばれ常に制御局が送受信を制御する方式である．制御局が従属局を順番に呼び出し，データ送信の許可を与える事を［f］と呼び，制御局がデータを送信するにあたって，従属局を呼出すことを［g］と呼ぶ．

［解答群］

ア．コンテンション　イ．HDLC手順
ウ．ポーリング
エ．セレクティング
オ．ポーリング／セレクティング
カ．CSMA／CD　キ．情報の転送
ク．回線の接続
ケ．データリンクの終結

解答

a－ク　b－キ　c－ケ　d－ア
e－オ　f－ウ　g－エ

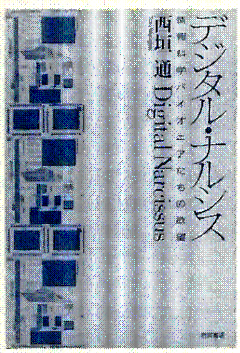


Book Review

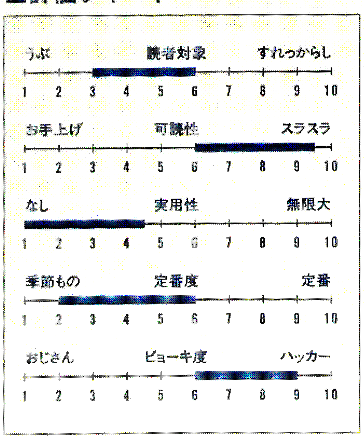
■中村正三郎

■デジタル・ナルシス

西垣通著
岩波書店
2,100円



■評価チャート



サイバー感覚への積極的な肯定

大変に魅力的なこじつけである．こじつけといっても悪い意味ではない．およそ世の中の文化論なり，社会論なり，その他もろもろの論であっても，所詮は都合のよいモデル作りであり，こじつけであると思ってしまえば，あとはその着想，手際，説得力が勝負になる．本書はその点，着想，手際，説得力いずれも刺戟的なこじつけであり，評者は大いにインスパイアされるものがあった．

副題に「情報科学パイオニアたちの欲望」とある．フォン・ノイマン，ウィーナーといった情報科学の天才たちが登場する．

全7章のうち，6章までの各章を一人ずつに割り当てている．目次を並べよう．なお，括弧内はその章で扱われる人物である．

「ゴーレムはよみがえった」(ノイマン)
「機械との恋に死す」(チューリング)
「階差に神はやどる」(バベッジ)
「通信路は断たれた」(シャノン)
「メタ・パターンを舞い踊る」
(ベイトソン)
「巨人は遅れてやってきた」
(ウィーナー)

著者は各章で，天才たちの喧伝されている業績の裏にあるもの，人間としての素顔をあぶり出す．

天才は一歩その歴史的業績から離れて実生活のレベルでながめると，多くの場合は変人と相場が決まっている．本書に登場する天才たちも例外ではない．まず，これらのエピソードが忘れられた歴史のページをめくっていく思いで興味深い．

読んでいると，誰それは，たしかにこういう育ちでこういう性格だからこそ，この偉業が達成できたに違いないと思ってしまう．手際がよく説得力がある．

7章「デジタル・ナルシス　機械は第三の性」は，6章までのまとめとして位置づけられる．もっといえば，6章までは7章の準備にすぎず，7章を書くために著者は周到にこじつけてきたと思える．評者がインスパイアされた部分は，この章が一番多い．

本書全体を，機械との共生感覚，サイバー感覚の積極的肯定が通奏低音として流れているのを感じた．もう少しタネあかし的に解読すれば，現代のようなサイバーパンク的世界が現出してしまっているのは，どうしてなのか．その鍵がパソコンやネットワークといった情報機械によるデジタルなナルシシズムにあると想定して，歴史を逆行すると情報科学のパイオニアたちの情熱の根底にもほら，といったところだろうか．

評者は哲学用語と引用されている哲学書を知らず，一部よくわからないところがあったが，理解できたつもりになっている．そして著者の考えとは違い，情報化社会・電子社会とそこに暮す人間像は，デジタル・ナルシスなどとそんな気取ったかっこいいものじゃないぞとも思った．

ともかくそんな想像(妄想？)が次から次へと生まれてくる，刺戟的な本である．

オブジェクト指向，Windowsが中心，92年の洋書動向

古田島義和（神田・書泉グランデ）

出版すればベストランキングの第１位をつねに獲得するWeinberg氏．前回，“コンサルタントの秘密（原題：The Secrets of Consulting）”がベストセラーになり，今回も“スーパーエンジニアへの道（原題：Becoming a Technical Leader）”がダントツのトップで，ベスト10の第２位と健闘しています．同氏のロングセラーになっているほかの作品は，“ライトついてますか（原題：Are your Lights on?）”，“システムづくりの人間学（原題：Rethinking Systems Analysis & Design）”，“計算機入力の人間学（原題：Humanized Input）”などで，いずれも訳本は共立出版，原本の出版社はDorset House社です．これから訳本として出版を期待したい同氏のオリジナルタイトルは，新刊の「Handbook of Walkthroughs, Inspections and Technical Reviews」，既刊では，「Software productivity」，「Exploring Requirements」，「General principles of Systems Design」，「Understanding the professional programmer」など，同氏の人気のすごさに脱帽です．

92年，最も注目される作品は，アスキーの「MINIXオペレーティングシステム」で知られるTanembaum氏が書いた，新しいOS「Amoeba」についての解説書「Modern Operating Systems」で，Prentice Hall（以下PH）から出版されます．本書は，分散OSについて書かれており，AmoebaとOSF/1やGNUで取り入れられたUNIX4.3BSDの上位互換“Mach”も取り上げているので，今後の分散OSの中心的存在となるでしょう．Machの解説書として，同社から「Design of the MACH Operating System」（Shienbrood著）が近日出版の予定です．また，Young氏のX Window System関連書の第３弾として，Open Look版「The X Window System:Programming and Applications with xt, Open Look Edition」が入荷，同氏の「Object-Oriented Programming and Design Using C++ and OSF/Motif」も出版予定です．同社からは，SPRACのマニュアル，SPRAC International「The SPARC Architecture Manual Ver8」も出版されました．

以前，本誌で紹介したMeyer氏のオブジェクト指向言語のニューバージョンEiffelが入荷し，「Eiffel:The Language」が評判通りの売れ行きを示しています．しかし，「Advances in Object Oriented Software Engineering」が出版未定となり，ユーザーのみなさんにはご迷惑をおかけしています．

Microsoft Press（以下MS Press）は，92年２月からWindows Ver.3.1対応の解説書を10点近く出版することを発表しました．「Windows 3.1 Tools」，「Windows 3.1 Guide to Programming」，「Windows 3.1 Ref Vol.1 Overview, Vol.2 Functions, Vol.3 Messages, Vol.4 Resources」の全４巻シリーズ，「Running Windows 3.1」，「M/S Windows 3.1:Quick Ref Guide」，「Windows 3.1 Companion」，「PenWindows 3.1 Programmer's Guide」です．すでに発売され好評なのが，PHの「Introduction to Microsoft Visual Basic」で，MS Pressでも92年１月に「Microsoft Visual Basic Workshop」を発売する予定で，Windowsの世界が幅広くなってきました．

■和書&洋書ベスト10

（書泉グランデ調べ：1991年11月１日～11月30日）

	和書	出版社
1	続・健康ソフトハウス物語	星雲社
2	スーパーエンジニアへの道	共立出版
3	UNIXオブザーバー1991.12月号	スペック
4	Z-MUSICシステム	ソフトバンク
5	NIFTY-Serve '91年度版	星雲社
6	カッコウはコンピュータに卵を産む(下)	草思社
7	カッコウはコンピュータに卵を産む(上)	草思社
8	UNIX4.3BSDの設計と実装アンサーブック	トッパン
9	笑うパソコンガール	ソフトバンク
10	Scheme入門	岩波書店

	洋書	出版社
1	Simple Book:An Introduction to Mqanagement of TCP/IP	Prentice Hall
2	C++ Programming Language 2/E	Addison Wesley
3	Undocumented DOS	Addison Wesley
4	Object-Oriented Modeling and Design	Prentice Hall
5	Standard C Library	Prentice Hall
6	ResEdit Complete	Addison Wesley
7	Motif Programming Manual V6	O'Reilly
8	Programming Perl	O'Reily
9	UNIX Network Programming	Prentice Hall
10	Annotated C++ Reference Manual	Addison Wesley

ワンラインNEWS
東芝はアクセス時間が12nsの1Mビット SRAMを開発，サンプル出荷を開始した．バイCOMSの構造と0.7ミクロンの微細技術を採用している．

★★★佐々木哲の★★★

U.S.プロダクツウォッチ

★不況下で業界再編成は続く

いまさらながらですが，不況が続いています．IBMは世界で30数万人余りの社員から２万人という大規模な人員削減を発表しています．機構の大幅な変更，分割まで予想されています．とくに，ハードウェアメーカーでの販売不振がきびしいようです．米国，日本だけではなく，世界のPCの生産基地，台湾でもかなりの数の会社の倒産が起きているようです．

この中で業界再編成は続いています，やはり一寸先は闇ですね．

★ファームウェアをフィールドアップデイト

ROMというものは，ハードウェアを新しく作るものにとってはなかなか厄介です．なぜ？　ソフトウェアならば，エンドユーザーに出荷後にバグや仕様変更などの問題が出てきても比較的容易にアップデートできますが，ROMに書き込まれたものはそうもいきません．PC本体のBIOS，ファームウェアのROMにはじまり，ハードディスクコントローラのROM，モデムのROMなどなど．エンドユーザーにROMを送りつけ，機械の蓋をあけてROMを差し替えさせるのはなかなか大変です．読者の方の中にもこの面倒な作業をされた方がいることでしょう．

最近，EEPROMやFlashメモリの普及にともない，フィールドで容易にファームウェアのアップデイトができる製品が出てきています．

DOS Extenderで名を知られるErgoのコンパクトなPC，The BrickはBIOSが書き換え可能になったもののひとつです．BIOS自身にDOS Extenderが入っているなど，野心的なパソコンなのですが，専用のアップデイトディスケットを使ってBIOSが新しいものに書き換えられるようになっています．メーカーは問題が出たときに，ユーザーに新しいBIOSが入ったフロッピーを送るだけでBIOSを差し替える手間などを省けます．

さらにモデムでは，このようなものがあります．Forval*1は独自の手法により物理層で21,600bps，V.32bisもサポートという高速モデムSA19200を販売しています．これもファームウェアのROMの内容を容易に書き換えられます．この場合は，モデムですからFDさえ必要ではありません．ユーザーはこのモデムと添付のソフトウェアでメーカーのサポートBBSにアクセスするだけです．その中でユーザーに見えない形でモデム本体のROMの内容がアップデートされます．

これからもこの傾向が増えていくことでしょう．ユーザーにとってとても使いやすく，またメーカーにとってもサポートの手間を省く，複雑なシステムの時代の要求です．

*1 Forval America Inc.
6985 Union Park Center, Suite 425, Midvale, Utah 84047
phone (801) 561-8080, fax (801) 561-8777

★RS-232Cの性能を絞り出すカード

このForvalのモデム自体，実に興味深いものですが，同時にForvalではマルチタスキンク環境などでの高速通信のための改良されたインターフェースを提案，提供しています．高速モデム，たとえばV.32モデムでは，V.42bisと組み合わせるとデータ転送速度は56kbpsを超えてしまいます．いくら今のPCが高機能になったとはいえ，マルチタスクなど負荷が大きい状態ではデータの取りこぼしなどがおきます．

ForvalのTurbo Interface*2と名付けられたものは，内蔵タイプのForvalモデムにはそのまま標準装備，外部式のものには別途インターフェースカードとして用意されています．あくまで今のRS-232Cというインターフェースのままで性能を絞り出すためのものです．

Turbo Interface Cardの構成を見てみると，８ビットXTバスのハーフサイズのカードに，モトローラMC68701のCPU，ナショナルセミコンダクタNS16C552 Dual UARTがのり，COMポートが２つ付く形になります．これで115.2Kbpsまでの安定した通信を保証します．既存の通信ソフトに対してもボード上のFIFOバッファが働き，仮想的に内部にFIFOをもったUARTとして見える形です．

ヘイズ社のESPのように高速での通信は専用のインターフェースというものもありますが，このようにあくまでも互換性にこだわった解決も捨てがたいもので

ワンラインNEWS
日本電気とオムロンは４ビットMCU用のファジー推論開発支援システム共同で開発し，販売を開始．このシステムを使いファジー推論システムの開発期間を短縮できる．

す.

＊2 Forval America Inc.
Turbo Interface Card

★その後のGeoWorks

以前ここでもふれたコンパクトでユーザーフレンドリーかつ強力なGUI，GeoWorksですが，その後の状況を少しふれます．

MOTIFのGUIを持ち，最初に多くの人に「真のWindows」としてインパクトを与えたGeoWorks，昨年暮れのGeoWorks Ensumble 1.0としてまず登場しました．使いものになるtrue WYSIWYGのワードプロセッサ，オブジェクト指向のグラフィックエディタを中心に，通信ソフトなどがそろい，買ってすぐに統合ソフトとして使えるものです．

version 1.0に続き約束通り，機能を強化したversion 1.2*3が現在販売されています．これはワードプロセッサ部にスペルチェッカーが付き，ポストスクリプトプリンタがサポートされるなどの改良が施されています．また，ゲームとしてソリテアに加えテトリスも含まれています．version 1.0の登録ユーザー全員にこれが送られています．

次に上位バージョンとして，GeoWorks Proが発表されました．統合ソフトとしての機能を充実させるために欠けていたスプレッドシートとして，なんとBorlandのQuattro Pro SEが含まれます．Lotus 1-2-3と上位互換でグラフィック指向のQuattro Proが，GeoWorksのGUI環境で動くのです．もちろん，印字にしてもGeoWorks OS自体に含まれているアウトラインフォントがそのまま使えます．GeoWorksのようなものを見ていると言葉の壁がある日本にいるのが本当に残念です．日本語が使えたら！

なお，GeoWorks自体は来年にはやっと待望のSDKが発表されるということです．これには当初いわれていたObject AssemblerのほかにCもサポートされるとか．また，GeoWorks Writer，GeoWorks Designer，GeoWorks Desktopとして，それぞれGeoWorks EnsumbleのGeoWrite，GeoDraw，残りのユーティリテイだけを含んだ安価なパッケージも用意されました．

＊3 GeoWorks Ensemble $199.00
GeoWorks Pro
GeoWorks Writer $69.99
GeoWorks Designer $69.99
GeoWorks Desktop $69.99
GeoWorks International
2150 Shattuck Ave., Berkley, CA 94704
phone (510) 489-8659

★VGA出力をテレビに

マルチメディア絡みというほどのものではないのですが，各種プログラムの表示，プリゼンテーションなど，PCの画面を一般のTV受像機に表示したりビデオにとるのはなかなか至難の技です．まだ米国でさえ，広く手に入らないものを紹介するのは気がひけるのですが，画期的な製品が台湾から出てきたようです．

Yuan Yuan Enterprise社*4のDEMO KEYは，PCのVGAビデオ出力をNTSCやPAL信号に変換してくれます．コンポジット出力とS-VHS端子が付いているので，そのままTVセットやVCR(VTR)に接続することができます．形状はちょうどパラレルポートにつけるXircomなどのLANアダプタほどのコンパクトなサイズです．これをパソコンのVGA出力端子につなげるだけです．

たとえば，日本のセガのテラドライブのようにVGAのうち，ゲームなどで使われる320×200の256色表示モードではNTSCの機器に出力できるものがありますが，これは通常のVGA画面も出力もできます．

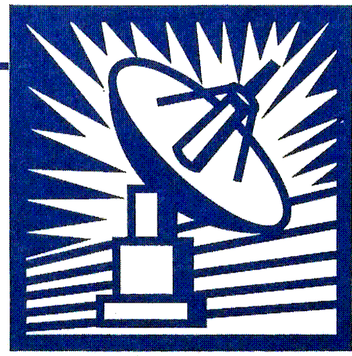
いままでのこの手の機器は外部の独立したボックスなど結構な価格でした(500～1,000ドル)．OEMむけの資料しか手元にないので定価はわかりませんが，いままでの数分の1になるでしょう．このような事情なので，以下にコンタクトしても1個単位でエンドユーザーが購入することはできません．あしからず．

＊4 Yuan Yuan Enterprise Co. Ltd.
No.54, Alley 2, Lane 76, Hsinyi Rd., Sec 6, Taipei, Taiwan, R.O.C.
Phone 886-2-759-0370, Fax 866-2-726-8845



Illustration : Masami Nozue

Illustration : NOBUKI FUKUHARA



西川和久の
希望的観測

OS/2 V2.0登場間近 はたして今後のニューウェーブとなるか！？

1 OS/2 V2.0の能力は……!? DOS環境を強力にサポート

最近注目を集めているOS/2 V2.0では，さまざまな環境が提供される．OS/2 1.x 16ビットVIO/PMアプリケーション，DOS，Windows，そして本命のOS/2 V2.0専用32ビットアプリケーション，これらの環境が同時に1台のマシンで使えてしまう凄いOSなのだ．これまでWindowsと比較して欠点とされていたDOSサポートも，MVDM（Multiple Virtual Dos Machine）になり，Windowsの上を行くグラフィックアプリケーションまでもがPMの窓上で作動する．さらに日本語版ではPC-DOS，JDOS，DOS/Vと，現在PS/55上で動くすべてのDOS環境をサポートするようだ．DR-DOSなど標準では対象外のDOSでさえフロッピーからブートすることによりちゃんと使えるカラクリも仕込まれている．早い話「なんでも来い!!」のOSなのだ．

またこれまでプログラムマネージャ，ファイルマネージャなど分離していたPM環境もオブジェクト指向のGUI「ワークプレイスシェル」に統一化され，ユーザーに優しく，MACを超える環境とうたっている．このようにスペックだけ見ていると「PC上GUI決定版」のように見えるOS/2 V2.0であるが，果たして真相はどうなのだろうか!?

2 わけがわからなくなりそうなβ版の嵐

“お堅い会社”のイメージをもつIBMとしては非常にめずらしく，英語版・日本語版共にかなり早い時期からβバージョンを各所へばらまいている．それも決まったバージョンではなく，とにかく最新版が出るたびに次から次から出してくる．βテスターも管理するだけでたいへんな状況なのだ．とくに，一番の売りのワークプレースシェルがバージョンごとにデザインが変わり，どれがどれだかわけがわからなくなってしまうほどである．

こうなったいきさつはいろいろ考えられるが，1つは「予定よりも出荷がはるかに遅れている」ことであろう．当初のスケジュールなら去年中には出荷する予定であった．ところが現在，今年の3月が英語版出荷開始の予定になっている．これも最終的にはどうなるかわからないが，米国の業界紙を読んでいると，OS/2 V2.0がこれ以上遅れると致命的であるという意見が大半を占め，年末が迫ったいま，もう何か月もないためプレッシャーは相当なものと思われる．

2つ目はやはりWindows NTの存在である．今年に入ってからIBMとMicrosoftの仲間割れは留まるところを知らず，エスカレートする一方なのだ．両者共，内部的にはかなり違うのだろうが，ユーザーから見れば似たりよったりのOSであり，すでにWindowsが離陸して

ワンラインNEWS
日本ビクターは日本電気のパソコン「PC-9801シリーズ」上で使用できる高品質のCG画像作成用ソフト「TRIMAGINE」をバージョンアップした．

いるため断然有利と見るMicrosoftと，DOS＋Windows環境をすべて取り込んでしまえばどうにでもなると考えるIBM．実際製品が出荷されるまでなんともいえないのが現状である．

3 パフォーマンスは!? Windows以上にマシンパワーを要求?

少し前のβバージョンになるが，最低作動条件ともいえるi80386SX/16MHz，5MB RAM，40MB HDDのPC互換機で英語版OS/2 V2.0を動かしてみた．Windowsの標準アプリケーションに相当する付属アプリケーションもいろいろなジャンルでかなりの本数が揃っている．とくに，IBMらしくスケジューラや表計算などビジネス色の強いアプリケーションがかなりの本数入っている．もちろんゲームもいろいろあったが，全体的にはオフィス指向の臭いがプンプンする仕上がりである（しかしあの有名な「猫」までもが標準装備とは……）．

さて肝心のパフォーマンスであるが，この程度のマシンスペックではかなり厳しかった．まず，40MBのハードディスクでは容量的にフルインストールできない．セットアップで必要ない機能を削り落とし，やっとなんとかインストールできた．MVDMやWindows，そしてPMアプリケーションなど一通り動かしてみたが，5MBのRAMではすぐにディスクスワップがはじまってしまい，そのままほっておくとハードディスクがオーバーフロー……そしてハングアップ．ディスクスワップがはじまるまでは，そこそこのスピードで動いていたが，全体として見ると同じマシンでWindowsを動かした時とくらべ明らかに遅い．

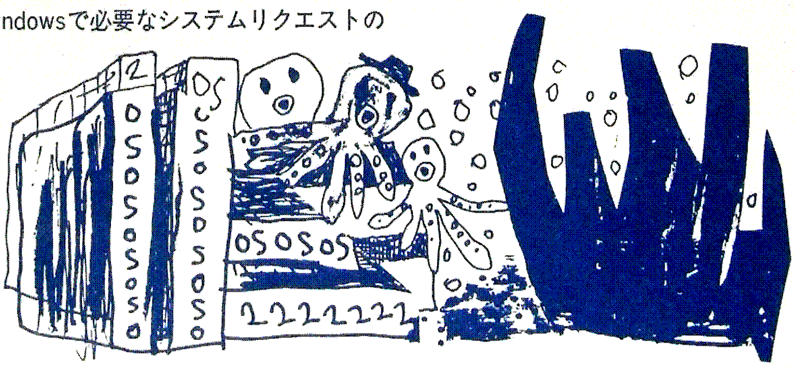
もちろんこれは製品版ではないので，今後チューンナップされるであろうが，βバージョンを触ったイメージ的には，Windowsで必要なシステムリクエストの少なくとも2倍以上のマシンが要求されるであろう．ストレスがなく快適に使えるとすれば，やはりi80486/33MHzフル装備の重装備マシンということになる（マア，Windowsも同じだけどね…… ^^;）．

4 互換機はどうなるの!? 日本版はOADGなら大丈夫?

一番心配なのが互換機でもOS/2 V2.0が使えるかどうかである．雑誌やネットなどいろいろな情報を眺めてみると，これは心配いらないようだ．Windowsに対抗するOSなのであるから，IBM純正マシンでしか使えないとなった日にはまったくお話にならない．

現在，IBMは互換機メーカーにβバージョンを配り作動チェックをかけている．すべての互換機でもOS/2 V2.0が使えるようかなり努力しているようだ．ただし，この話は英語版のことであって，日本語版はどうなるかまだわからない．少なくともOADGメーカー内では動くようにするだろうが，グラフィックアダプタの問題など解決しなければならない問題点が山のように残っている．またMicrosoftとIBMが喧嘩している間は，非PC系のマシンに日本語OS/2 V2.0が乗ることは当面ないであろう．

これはOS/2に限った話ではなく，Windows 3.1も出荷が遅れ，やっと姿が見えはじめたWindows NTやその他のOS……今後のOS争奪戦は留まるところを知らない．DOSでさえも，すでにV6.0の話がもち上がり，さらにDR-DOSと……まあよくここまで混乱させてくれるものだ．とくに，日本語版はその特殊性やタイムラグが発生するため余計話がややこしくなる．最後にユーザーがババを引かないよう，いまは祈るだけである．



コンピュータの法律

他人にプログラムを作成させた場合の著作権の帰属　その3

熊谷信太郎（弁護士）

■ソフトハウスに委託した場合の著作権

今回は，ソフトハウスやメーカーにプログラム作成を委託した場合の著作権の帰属について考えてみたいと思います．

ソフトハウスやメーカーにプログラム作成を委託した場合には，プログラム著作権は発注者にではなく，実際にプログラムを作成したソフトハウスやメーカーが取得します．前回までに検討したように，感覚と技術を駆使し，自らの創意と手法によりプログラムを創作したのは発注者ではなくソフトハウスやメーカーだからです．

もっとも著作権法は，「法人等の発意に基づきその法人等の業務に従事する者が，職務上作成するプログラムの著作物の著作者は……その法人等とする」*1と定めているので，会社がソフトハウスやメーカーにプログラム作成を委託する行為が，この規定にあたるならば，発注者である会社に著作権が帰属することになります．

しかし，「法人等の業務に従事する者」というのは，人だけを指し，会社などの法人は含まないと考えられていますので，会社の従業員などはこれに含まれますが，下請け企業やメーカー，ソフトハウスなどの取引先はこの規定にはあたりません．

したがって，著作権法の規定上からは，メーカーやソフトハウスがプログラムの著作権を取得することになります．

■著作権と著作者人格権

しかし，メーカーやソフトハウスにプログラムの作成を発注する際に，発注者が著作権を取得するという契約を結ぶことは可能ですし，現に，そのような契約は多くなされています．

その場合，法律的には著作権はいったんはメーカーやソフトハウスのところで発生し，それが契約により発注者のところに移転するということに注意しなければなりません．

これはどういうことかというと，著作者は通常の著作権と著作者人格権の2つの権利をもっていますが*2，このうち複製権，上映権などの通常の著作権は譲渡可能な権利とされていますから，契約により発注者のところに移転することに問題はありません．

これに対し，公表権，氏名表示権，同一性保持権の3つから構成される著作者人格権は，その性質上著作権者の一身専属的な権利であり，譲渡できないものと考えられているので，たとえ譲渡契約をしても，譲渡の効力を生じません．

まず，公表権とは，まだ発表されていない著作物を公衆に提供し，または提示する権利です*3．これについては，未公表の著作物を譲渡した場合には，公表について同意したものと推定されていますから，発注者がメーカーやソフトハウスからプログラムを買い取る場合にはあまり問題を生じません．

次に，氏名表示権とは，著作物の原作品に氏名などを表示し，あるいは表示しない権利です．

これについては，著作物の利用の目的および態様に照らし，著作者が創作者であることを主張する利益を害するおそれがないと認められるときは，公正な慣行に反しない限り，著作者名の表示を省略できるとされていますので，メーカーやソフトハウスにプログラムを発注した場合には，さほど問題にならないと思われます．

最も問題になると思われるのは，同一性保持権です．同一性保持権とは，著作者の意に反して変更，切除，その他の改変を受けないという権利のことですが，それは前述のように一身専属権ですから，譲渡契約によっては発注者に移転できません．したがって，メーカーやソフトハウスにプログラムを発注した場合には，メーカーやソフトハウスだけが同一性保持権を有することになり，発注者としては，メーカーやソフトハウスにいちいち承諾を得ないとプログラムの改変ができないことになるわけです．

しかし，それでは発注者としてはバージョンアップなどの際に，たいへん

困ることが予想されます．そこで，その場合の対策として，発注者がメーカーやソフトハウスとプログラムの譲渡契約を締結する際に，メーカーやソフトハウスはプログラムの著作者人格権を行使しないという特約を結んでおくことが考えられます．著作者人格権を譲渡する契約は無効ですが，著作者人格権を行使しないという契約は一身専属性に反しませんから，有効と考えられているのです．

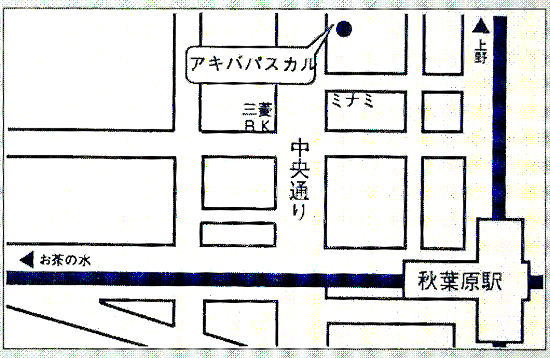
＊1　著作権法第15条第2項
＊2　著作権法第17条第1項
＊3　著作権法第18条第1項

アキバ短信

アキバパスカル

秋葉原Macショップ事情

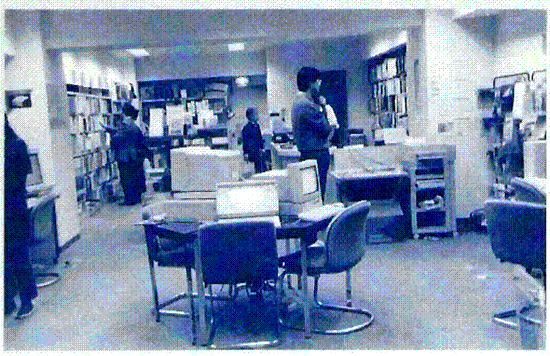
営業時間　11:00〜19:00
定休日　毎週水曜日
TEL　03-5256-8001



読者の皆さんは，暮れのボーナスで何を買ったのだろうか？　念願の486マシンか，はたまたMO，あるいはHSTだろうか．すでに購入したマシンのローンを払うだけだよという人には，まったく目の毒，気の毒だが，昨年秋からは，メーカー側のボーナス商戦をあてこんだ新製品発表が目白押しだった．なかでもMacの新製品は，プログラマのみならず，一般パソコンユーザー，あるいはユーザー未満の人びとの耳目を集めることとなったのは，まだ記憶に新しい．

数年前まではMacといえば高価なパソコンの代名詞で，一部のミュージシャンやデザイナーに熱狂的に愛されているという話を耳にする程度であった．Macを買うには清水の舞台から飛び降りる決意は必要なくとも，ローンの保証人が必要だった．それが，いまや一般の雑誌にワープロやノートパソコンと並んでPowerBookが紹介されるところまで大衆化した．「SimAntやりたいからClassic II買っちゃおうかな」というご時勢である(何を隠そう，このアキバ短信の地図も編集者が片手間にMacで描いているのだ)．

秋葉原でもMacintoshを取り扱う店が急速に増えている．LAOXやソフマップ，ツクモ電機などの大型店はもちろん，あのステップもMacには自社でサポート部門を設けるほどの力の入れよう．老舗のT-ZONEも売り場を広げ，イケショップは新たに「お茶の水店」を開設した．リッツ，ハイパークラフト，マックプランナー，ナビック，ZEROなど専門ショップも着実に数が増えてきた．



パスカルは東京ユニテック(株)の経営するMac専門店で，昨年末で3周年を迎えた．Pascal Fellowという会員制度を設けて会員に3%のディスカウントを行ったり，ファクシミリで最新の製品情報を送ってくれたり，取扱い商品の品目表を作ったりと，ユニークなサービスを展開している．もちろん，通信販売もOK．改装のために，一時店舗を移転していたが，現在は新装なって，マヤ電機の隣の真新しいビルの2階で営業中だ．

中央通りをはさんで向かい側から，脇道に入った一画には，ソフマップの中古Mac専門店や，ASK，ハローなど1階に林檎の看板を出している店も多いので，ちょっと足を伸ばしてみてもおもしろい．アキバパスカルに限ったことではないが，Macの専門店は秋葉原駅から離れた場所，しかも雑居ビルの2階，3階といったところが多い．はじめての人はMac専門誌の広告などで場所を確認してから出かけたほうがよさそうだ．

ワンラインNEWS
東芝は高速印字が可能なワープロ「ルポJW95HV」を発売した．160ドットと幅の広い印字ヘッドを使用し印字速度毎秒230字を実現している．

●プログラマの仕事場 11

Programmer's Working Space

ビットマップ・ブラザース

欧州No.1のゲームソフト「GODS」を生んだイギリスのプログラマ・チーム

今回は，英国のスター・プログラマ・チーム，ビットマップ・ブラザースを紹介する．彼らは，欧米のゲーム市場でいま人気の，ヘラクレス伝説をベースにしたアドベンチャーゲーム「GODS」(Amiga, IBM PC版)の制作チームである．「GODS」の日本での発売決定を機に，日本の市場視察をかねて来日した，ビットマップ・ブラザースのエリック・マシューズ氏と彼らの所属するレネゲード社のマネジメントディレクターであるトム・ワトソン氏にお話をうかがった．

音楽産業＋ゲームプログラマ＝マルチメディア

去る11月，ヨーロッパで人気のゲーム「GODS」の日本での発売が発表された．テトリスやシムシティを筆頭に，いまや海外で開発されたコンピュータゲームが日本で出ることそれ自体は，もはやめずらしいことではない．ただ，この発表が従来と違ったのは，日本側がセガや任天堂といった大手のゲーム機会社でも，ゲームソフト会社でもなく，PCM completeという聞き慣れない名前であったことだ．

音楽業界の事情に詳しい人ならピンときたかもしれない．PCM completeは，筋肉少女帯やカブキロックスを抱える音楽プロダクション，プレスクラブミュージックを母体とし，レコード会社や著作権管理会社を含めたグループの総称である．そのPCM completeが，これまたイギリスのレコード会社＋音楽プロダクションであるリズムキング社と関わりの深い，ビットマップ・ブラザースと提携し，彼らの作ったゲームをメガドライブ，PCエンジン，PC−9800シリーズ用に発売するというのである．ビットマップ・ブラザースは，欧米においてスター・プログラマ・チームとして非常に有名な存在であるという．

音楽プロダクションとプログラマ，ちょっと耳にしただけでは，何の関連もなさそうな取り合わせだが，トム・ワトソン氏はリズムキング社とレネゲード社，ビットマップ・ブラザースの関係について，次のように語っている．

「もともと89年にさかのぼる話になりますが，当時私はミラーソフトのマーケティングを担当していました．そのころ，ビットマップ・ブラザースに所属しておりまして，担当のゲーム「ゼノン2」製作中に，リズムキングのアーティストである，ボム・ザ・ベースの曲を使いたいということで，リズムキング社と話をしたのです．これからの娯楽産業に関する考え方が一致した点がありまして，そういった経緯で設立されたのがレネゲード社です．レネゲード社を設立したのは，これからはマルチメディアが盛んになるであろうと予測したからです．とくに，その中では音楽が重要視されてくるであろうというわけです．メディアをミックス

する際に出版や音楽の著作権などを管理するためのケースとして，つまり，今後マルチメディアの中で著作権などの問題が生じてくる場合でも，柔軟に対応していけるような会社を作ろうということだったのです.」

「イギリスの市場でも，マルチメディアというコンセプトは注目されていますが，その実体がまだつかめないという印象がもたれています．CDというメディアについては，音楽市場ですでに評価されていますから，ゲーム市場，もしくはホームコンピュータ市場においても，AmigaがCD-ROMを搭載したことなどから注目が集まっています．そういった意味でわれわれも参入しているわけです．われわれと提携しているリズムキングにはポップアートという映像関連の会社があります．これから，音楽・映像・コンピュータの関係はますます重要視されていくでしょう.」

「開発チーム」は英でも新しい開発形態

ビットマップ・ブラザースというのは，ソフトウェア会社ではなく，開発チームの名称であるという．彼らのいう「開発チーム」とは，いわゆるソフトウェアハウスの中で，複数のグループが作業を進めるのとは，少々異なるようだ．ビットマップ・ブラザースでゲーム・デザイナーをつとめるエリック・マシューズ氏は，

「ビットマップ・ブラザースは企画から完成まですべてを行います．それを販売し，著作権を取り，経営を管理するのがワトスン氏です．レネゲード社は基本的にパブリッシャーで，資本的にはリズムキングとビットマップ・ブラザースが半分ずつ持っています．ビットマップ・ブラザースが制作したものは，レネゲード社が買い取り――というと少し変ですが，販売することになっています」と語ってくれた．

つまり，ミュージシャンがレコード会社からレコードを出すように，ビットマップ・ブラザースというプログラマ・チームが，レネゲード社というブランドから，ゲームソフトを世に送り出すということらしい．そうはいっても，営業サイドと，制作者の間には通常何らかの意見の食い違いというのが発生しがちなものだが，マシューズ氏はレネゲード社の芸術的な方面での取締役もかねているため，そういった問題は発生しないという．

では，ビットマップ・ブラザースには，どんな人たちが集まっているのだろう．

「ビットマップ・ブラザースは全部で12名います．3～4人ずつ，3ないし4つのチームに分かれて，つねに別々の違う製品を開発しています．ひとつのチームの中に，それぞれプログラマ，デザイナー，ミュージシャン，映像関係のグラフィックの担当者がいます．分かれているといっても，2つのプロジェクトにひとりの同じデザイナーがつくという重複した形態もありますが，基本的には各チームに核となるメンバーが3～4人いることになります.」

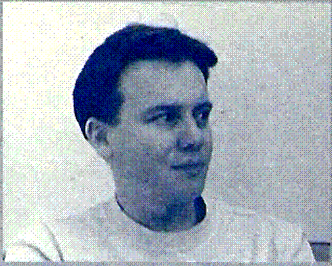
「いままでは，プログラマという人がゲームを作る上で一番重要な役割だといわれていましたが，私は少し違った見方をしています．デザイナーや映像，音楽担当者，そしてプログラマにも平等に責任があるといいますか，それぞれの役割を果たさなくてはいけないと思います.」

Personality

トム・ワトソン
THOMAS(TOM) WATSON
レネゲード社マネージング・ディレクター．大学卒業後，イギリスの大手コンピュータ雑誌出版社に入社．3年後，イギリスのテレコムでマーケティング・ディレクターとしてソフトウェア・オペレーションに従事した．その後，2年間ミラーソフト社でマーケティング・ディレクターとして従事した後に，レネゲード社創設のため離職．



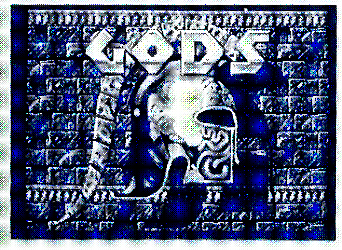
エリック・マシューズ
ERIC MATTEWS
学生時代からゲーム産業に携わる．Chelsea School of Art卒業後，Royal Academyのマスターコースへ進学，そのころからコンピュータゲームのグラフィックを手掛け，卒業後はフリーでコンピュータ・グラフィック制作を行っていた．ビットマップ・ブラザースが成長してきた段階で，グラフィック制作をやめ，現在はゲーム・デザインを専門に手掛けている．



GODS

ギリシャ神話のヘラクレス伝説をベースにしたゲーム．敵のキャラクターと対決しながら永遠の生命を勝ちとるアクション+ロール・プレイングゲーム．AI制御によって，敵のキャラクターが違う攻撃をしかけてきたり，戦略を立ててプレーヤーのゲームの

進行を妨げるなどの要素が盛り込まれており，繰り返しプレーしても楽しめるという特徴がある．ヨーロッパ市場では，ゲームチャートの上位にランキングされている．日本では，メガドライブ，PC-エンジン，FM-TOWNS，PC-9800シリーズなどへの移植が行われる予定で，7月から順次発売される．問い合わせ先は（株）セティオン（TEL. 03-5273-4094）．



ヨーロッパでは Atari, Amigaが人気

ビットマップ・ブラザースの「GODS」は現在，Amiga版，IBM PC版がリリースされている．そこで，開発のベースとなるマシンや，開発環境についてたずねてみた．

「一番よく開発に使うマシンはAtariのSTですが，これを使う理由は4つあります．まず第一に市場に出された一番最初の16ビットマシンであるということ，第二にプログラマがそのアーキテクチャを好んできたこと．第三は制限の多いマシンなので，その中で楽しみの多いゲームを作ることができれば，ほかのマシンに移植しても，より楽しみが生まれるのではないかということ，そして4点めとしては，ヨーロッパ市場ではAmigaとAtariが主要マシンなのですが，OSをくらべるとAmigaのほうがずっと嫌なOS（フロッピーを何度も交換しなければならないなど）だからということです．ただ，これからは少しずつ変わっていきそうです．」

「今後は国際的に活躍していきたいと望んでいるのですが，活躍していく上での問題となるのは，日本とアメリカのマーケットは，ヨーロッパのマーケットとあまりにも違う部分があるということです．いま，ヨーロッパではPCのマーケットはAmigaにくらべると5分の1の規模しかありません．Atariとくらべると4分の1です．しかし，日本やアメリカでは98，IBM PCが全盛です．そのために，選択の悩みを抱えることになるのです．方向性としては，PCの能力を発揮するようなゲームはPCで作り，それ以外のものはAmiga，Atariなどで作っていきたいと考えています．」（マシューズ氏）

失業率の高さから やむをえずプログラマになる人も

日本ではこのところ，プログラマをはじめソフトウェア技術者の不足が叫ばれている．優秀な技術者はどこも喉から手がでるほど欲しい状況だが，イギリスではそのような問題は起きていないのだろうか．

「人数的には技術的に優れたプログラマは多いので，人材の不足は感じません．その背景としては，専門的な知識がない人でも，弟子入りのような形で入って，後輩を育てるということが積極的に行われているので，つねに人材が出てくるということがあります．イギリスが一番よいとはいいませんが，恵まれた環境にあります．」

「ビジネス用のプログラムを作成するプロフェッショナルのプログラマと，娯楽用のコンピュータのプログラマとを比べてみますと，ゲームのプログラマは専門のコンピュータ教育を受けていなくても，好奇心の旺盛な人が多いので，自分から率先してその道に入る人が多いのです．いわゆるビジネス用のプログラマについては，イギリスでは失業率が高いので，職業が欲しいために半分はやむをえず，そういった職業についてしまう人もいます．そういう意味では，ゲームプログラム業界は，よい人材が出てくる可能性が高いといえるでしょう．」（マシューズ氏）

失業率が高いためにやむをえずプログラマになる人がいるというあたりはやはりイギリスなりの事情を反映しているのだろう．マシューズ氏は続けて，アメリカとヨーロッパのプログラマの違いを次のように語ってくれた．

「また，アメリカとイギリス，フランス，ドイツのプログラマの違いについていいますと，アメリカでプログラマをやっている方々や製品を見ていますと，製品としてはたしかによいのですが，非常に強力なコンピュータを使わないと実行できないものが多いと思います．ヨーロッパでは，限られた制限のある枠の中で最大限の努力をするという風潮があります．そういった意味で，ヨーロッパとアメリカとでは状況が違います．ヨーロッパ市場のプログラマは，プログラマという職種の集団自体がタイトで連絡関係も密です．お互いに切磋琢磨したり，特定の機械を使いこなすためのいいライバル関係があります．」

おもしろいゲームはどんな国でも受け入れられる．そういう意味で，ソフトウェアに国境はないといえるかもしれない．海外から日本市場へ，日本から海外へ，ソフトウェアの国際化はいっそう進んでいくのではないか，そんなことを予感させるようなインタビューだった．

ワンラインNEWS
任天堂は16ビット家庭用ゲーム機「スーパーファミコン」用の高速演算用処理チップを開発し，一部のメーカーにサンプル出荷を開始した．

information・page

●イベント

■1月22日(水)～1月25日(土)
第21回インターネプコン・ジャパン'92
エレクトロニクス製造技術展
第9回エレクトロテスト・ジャパン'92
エレクトロニクス試験・検査機器の専門展
第7回ADEEジャパン'92
IC/PCB設計のためのCAD/CAE関連機器展
会場：日本コンベンションセンター(幕張メッセ)／入場料：3,000円
問い合わせ先：カーナーズエクスポジションジャパン　TEL 03-3349-8501

■1月29日(水)～1月31日(金)
PAGE92
エレクトロニックプリプレス、コンピュータパブリッシングなどの関連製品と最新技術の展示会。
会場：サンシャインシティコンベンションセンターTOKYO(東京・池袋)
入場料：500円
問い合わせ先：(社)日本印刷技術協会　TEL 03-3384-3411

■2月20日(木)～2月22日(土)
MACWORLD Expo/TOKYO
Macintosh関連製品の展示とコンファレンスが行われる。
会場：日本コンベンションセンター(幕張メッセ)
入場料：【前売り】33,000円(コンファレンス)、2,000円(展示会・3日間通し)
【当　日】38,000円(コンファレンス)、1,500円(展示会)
問い合わせ先：日本工業新聞社事業部MACWORLD Expo/Tokyo事務局
TEL 03-3292-3561

■2月26日(水)～2月28日(金)
ボード・コンピュータ'92
CPUボード、メモリボードなどの産業用ボード・コンピュータの専門展示会。
会場：サンシャインシティコンベンションセンターTOKYO(東京・池袋)
入場料：1,000円
問い合わせ先：(社)日本能率協会　TEL 03-3434-1377

■2月26日(水)～2月28日(金)
LAN Expo '92
LAN(ローカルエリアネットワーク)関連製品の総合展示会。
会場内のブースをLANで結び、LANを直接体験することができる。
会場：東京国際見本市会場(東京・晴海)／入場料：1,000円
問い合わせ先：(株)マーコム・インターナショナルLAN Expo事務局
TEL 03-3403-8515

●セミナー

■2月7日(金)
次世代FAソフトウェア開発の方向性と標準化の動向
国際標準化を検討しているISO/TC184の各サブコミッティの活動を中心に、今後の生産システムの鍵となる技術動向と標準化活動の位置づけについて講演。
会場：中央大学駿河台記念館(東京・神田)
受講料：28,000円(会員)、42,000円(非会員)
問い合わせ先：(社)精密工学会　TEL 03-3362-1979

■1月21日(火)～1月23日(木)
オブジェクト指向設計とC++セミナー
オブジェクト指向技術によるC++プログラミングを体験学習する。
会場：[illegible]の門パストラル(東京・港区)
受講料：190,000円(3名以上、2講座受講者に割引あり)
問い合わせ先：ソフトウェアテクノロジーインターナショナル
TEL 03-5689-3536

■2月19日(水)～2月20日(木)
オブジェクトオリエンテッドテクノロジー入門セミナー
オブジェクト指向による分析設計プログラミングの長所から短所までを解説する。
会場：麹町会館(東京・千代田区)／受講料：96,000円(3名以上割引あり)
問い合わせ先：ソフトウェアテクノロジーインターナショナル
TEL 03-5689-3536

Microsoft University

■1月27日(月)～1月31日(金)、2月24日(月)～2月28日(金)
MS OS/2 Applications Programming
OS/2アプリケーション開発のための上級コース。
受講料：180,000円(5日間)

■1月27日(月)～1月31日(金)、2月17日(月)～2月21日(金)
Microsoft Windows Applications Programming
Windowsアプリケーション開発のための上級コース。
受講料：180,000円(5日間)

■1月27日(月)～1月31日(金)
Microsoft LAN Manager Applications Programming
LANマネージャアプリケーション開発のための上級コース。
受講料：180,000円(5日間)

■1月27日(月)～1月29日(水)、2月19日(水)～2月21日(金)
C言語応用
MS-C 6.0を使ってC言語の高度な技術を学ぶ実習中心のコース。
受講料：90,000円(3日間)

■2月3日(月)～2月7日(金)、2月24日(月)～2月28日(金)
Microsoft Windows Programming Environment
Windowsアプリケーション開発のための入門コース。
受講料：180,000円(5日間)

■2月17日(月)～2月18日(火)
C言語入門
QuickCを使ったC言語の入門コース。
受講料：60,000円(2日間)

■2月17日(月)～2月21日(金)
Microsoft LAN Manager Programming Environment
LANマネージャアプリケーション開発のための入門コース。
受講料：150,000円(5日間)

■2月25日(火)～2月28日(金)
MS OS/2 Device Drivers
OS/2デバイスドライバの機能と構造を解説する。
受講料：150,000円(4日間)

■2月3日(月)～2月7日(金)
MS OS/2 Presentation Manager Applications Programming
OS/2 PMアプリケーション開発のための上級コース。
受講料：180,000円(5日間)

Microsoft Universityは、OS/2、Windowsアプリケーション開発者のために定期的に開催されているセミナーです。
問い合わせ先：富士ソフトウエア(株)品川教育センター　TEL 03-5461-8296

NEWS & NEW PRODUCTS

ボーランド，98用Borland C++ & Application Frameworksを発売

ボーランド㈱（TEL 03-3486-1400）は，高速なコンパイルスピードと大規模なアプリケーション開発に適したプロテクトモードのコンパイラを備え，MS-DOSとWindowsアプリケーションをCとC++で開発することのできる強力な開発環境，PC-9800シリーズ用「Borland C++ & Application Frameworks 2.0」を発売した．これは，既存のBorland C++ 2.0にApplication Frameworksを組み合わせた製品で，Application Frameworksは，Windows 3.0用のObject WindowsとMS-DOS用のTurbo Visionの2種類のフレームワークからなり，必要なプラットフォームに合わせて選ぶことができる．どちらもマウス対応のオーバーラップマルチウィンドウやメニュー，スクロールバー，ダイアログボックス，エディタなどの機能を提供する．ほかに，Turbo Driveを使用した大規模なプログラム開発を可能にするプロテクトモードコンパイラ(BCX.EXE, BCCX.EXE)，MS-DOSとWindows 3.0に対応した統合環境デバッガ(Turbo Debugger 2.5)，MASM互換の高速アセンブラ(Turbo Assembler 2.5)，統合環境型プロファイラ(Turbo Profiler 1.1)，メニューやダイアログボックスなどの管理，編集作成ができるWindows 3.0用のリソースエディタ(Resource Workshop)を備えている．価格は98,000円．

■動作環境

対応機種：NEC PC-9800シリーズ(ハイレゾモード，LTを除く)，EPSON PC-286/386シリーズ／必要メモリ：640KB以上(Turbo Driveの使用には1MBの拡張メモリが必要)／OS：MS-DOS Ver.3.1以上，MS-Windows Ver.3.0以上／周辺機器：ハードディスク(必須)，マウス，EMS対応

テックソフト&サービス，32ビットプロテクトモード開発ツールを発売

テックソフト＆サービス㈱（コンピュータ事業部パッケージ販売部：TEL 0474-23-1322）は，80386，80486CPUのプロテクトモード専用のDOSエクステンダ開発ツール「386｜TOOLBOX Version 4.0」を発売する．「386｜TOOLBOX」は，米国PharLap社の開発した開発ツールキットで，膨大な拡張メモリをフラットな領域としてユーザープログラムに提供し，MS-DOSの制限である640KBのメモリ制限や64KBのセグメントの壁を越え，巨大なメモリを必要とするプログラムの作成を可能とするDOSエクステンダ，8086から80486，8087，80287，80387のすべての命令を使用することができるアセンブラ，アセンブラやHIGH C386などで生成された32ビットオブジェクトとMASMなどのOMF-86互換の16ビットオブジェクトとのリンクも可能なリンカのほか，ライブラリアン，デバッガ，仮想記憶マネージャとそれらの環境カスタマイズユーティリティを装備している．発売は2月の予定．対応機種は，CPUに80386，80386SX，80486を搭載したPC-9800，H-98シリーズ，EPSON PC-386シリーズ，IBM PC/AT互換機など．価格は198,000円で，開発された製品を販売するためにはランタイムライセンスを購入する必要がある．

CSK，SUN，HP用LISPの販売・サポートを開始

㈱CSK（TEL 03-3344-1811）は，米国ルーシッド社の開発したLucid Common LispのSUN，HP版の販売，サポートを開始した．従来は，「Sun Common Lisp」，「HP Common Lisp」として，Sun, HPにOEM供給されていたが，米国での版権，販売権がルーシッド社に移行したことに伴い，日本国内でも日本サン・マイクロシステムズ，横河・ヒューレット・パッカード社からCSKへの移行協議が成立し，同社が販売・サポートを行うことになった．これにより，同社では「Lucid Common Lisp/Sun」，「Lucid Common Lisp/HP」として出荷を開始し，既存のユーザーに対しては，技術サ

ワンラインNEWS
ニューテックはワークステーション上でCD-ROMが簡単に作成できるシステム「CD Writer-Sun」を発売した．同製品を使うことでCD-ROM作成の一環作業が可能になる．

NEWS & NEW PRODUCTS

ポート，バージョンアップを含む新しいサポート契約への移行を推進していく．また同時に，ソニーのRISC NEWS用の最新バージョン4.0「Lucid Common Lisp/RISC NEWS」の販売・サポートも開始した．

毎日コミュニケーションズ，SunOS版「NAWK | AWKCC」を発売

㈱毎日コミュニケーションズ出版事業部（販売促進課：TEL 03-3211-2568）は，SunOS版パターン処理言語インタープリタ，コンパイラ「NAWK | AWKCC Rel 2.0」を発売した．「NAWK」（日本語AWK）は，パターン処理言語と呼ばれるインタープリタの処理系で，テキストファイルの中からあるパターンを見つけ，それに対応したアクションを行う．ほとんどのプログラムが2，3行からなるため，プログラマ以外でも簡単に使うことができる．「AWKCC」は，NAWKで記述されたプログラムをC言語のプログラムに変換するコンパイラで，変換されたプログラムは，NAWKのプログラムとまったく同じ内容をより速く実行することができる．価格は98,000円．

富士通，FM Rシリーズ用「日本語MS-DOS V5.0」を発売

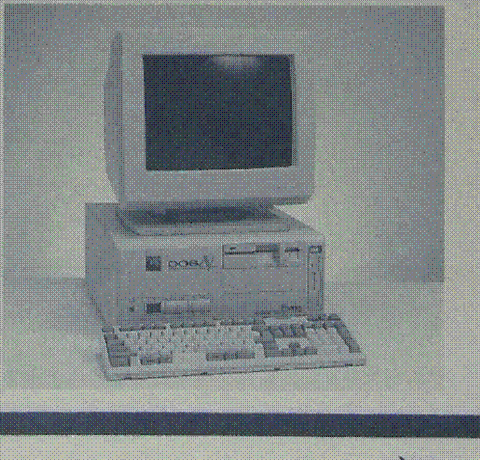
富士通㈱（FMインフォメーションサービス：TEL 03-3646-0816）は，パーソナルコンピュータFM Rシリーズ用OSの新バージョン「日本語MS-DOS V5.0」を発売する．「日本語MS-DOS V5.0」は，カラーおよびグラフィックを使用したビジュアルシェルの提供により，ファイル操作やコマンドの起動をマウスまたはキーボードから簡単に行うことができ，メモリ管理機能の強化により，HMA（High Memory Area）にMS-DOSシステムの一部を移行，EMM386が利用可能な機種では，OAK（かな漢字変換機能）を拡張メモリ上に展開することにより，FM R-80HL1の場合で，ユーザーメモリを最大約60KB拡大することができる．MS-DOSシェルのプログラムセレクタにアプリケーションを事前に登録することにより，複数のアプリケーションを同時に起動し，切り替え運用が可能になる．出荷は2月初旬の予定．価格は，アプリケーションプログラム利用ユーザー向けの「日本語MS-DOS V5.0（基本機能）」が12,800円，アプリケーションプログラム開発ユーザー向け「同（拡張機能）」が10,000円．

関西電機，DOS/V対応マシン「新P1シリーズ」を発売

関西電機㈱プラスワン（TEL 03-5687-6912）は，既存のAT仕様マシン「PLUS ONEシリーズ」を全面的にDOS/V仕様に変更した「新PLUS ONEシリーズ」3モデルを発売した．新たに，3.5インチドライブと日本語キーボード，バスマウスを標準添付したほか，LSIチップの変更，ハードディスクのアクセス速度の向上などをはかった．

■48AG-STD/48AG-H120（写真）

プロセッサ：80486（33MHz）／メモリ：4MB（最大64MBまで拡張可能）／外部記憶装置：3.5インチFDD×1・5インチFDD×1（STD），3.5インチFDD×1・5インチFDD×1・120MB HDD×1（H120）／価格：598,000円（STD），798,000円（H120）



NEWS & NEW PRODUCTS

■38HM-STD/38HM-H120
プロセッサ：80386（40MHz）／メモリ：4MB（最大32MBまで拡張可能）／外部記憶装置：3.5インチFDD×1・5インチFDD×1（STD），3.5インチFDD×1・5インチFDD×1・120MB HDD×1（H120）／価格：448,000円（STD），648,000円（H120）

■38FV-STD/38F-H40
プロセッサ：80386（25MHz）／メモリ：4MB（最大16MBまで拡張可能）／外部記憶装置：3.5インチFDD×1・5インチFDD×1（STD），3.5インチFDD×1・5インチFDD×1・40MB HDD×1（H40）／価格：268,000円（STD），348,000円（H40）

エプソン，32ビットノートパソコンと統合ソフト「HARMONY」をセット販売

セイコーエプソン㈱，エプソン販売㈱（エプソンPCインフォメーションセンター：TEL 03-3377-3531）は，同社の32ビットノートパソコン「PC-386NOTE AE」とロータス㈱から発売されたスーパー統合ソフト「HARMONY」のセット製品を発売した．「HARMONY」は，ロータス㈱と㈱ジャストシステムが開発した表計算，データベース，グラフ，通信，メモサービスの機能に，一太郎をベースにした日本語ワープロ機能を一体化した統合ソフトで，40MBのハードディスクパックにあらかじめインストールされている．価格は426,000円．

富士通・PFU，UNIXコンピュータ「Aファミリー」を強化

富士通㈱（システム第二販売推進部：TEL 03-3216-8004），㈱PFU（東京本社営業支援部：TEL 03-3484-0311）は，UNIXコンピュータ「Aファミリー」に，コンパクトAの最上位機種「A-120」とエントリモデル「A-MICRO」を追加，発売を開始する．「A-120」は，MC68040（33MHz）の採用により，従来機A-100の約1.3倍の性能向上をはかり，最大112MBのメモリを搭載することができる．出荷は5月の予定で，価格は14,800,000円より．「A-MICRO」は，CPUにMC68030（22MHz）を採用し，小型・省スペース，ハイコストパフォーマンスを実現している．価格は1,950,000円より．また同時に「Aファミリー」全体についても，32MBメモリの提供，小型磁気ディスク装置，事務処理用高生産ツール「EPOACE/UX」のサポートなど，ハードウェア，ソフトウェアの強化を実施した．

メルコ，簡単で経済的なプリンタネットワークを発売

㈱メルコ（メルコインフォメーションセンター：TEL 052-251-8365）は，低コストでプリンタネットワークを構築することのできる製品群を発表した．製品には，2入力1出力のプリンタネットワーク専用自動切替機「トランシーバー」（NTR-5（5m）：4,300円，NTR-20（20m）：5,800円），8入力1出力の「マルチポートトランシーバー」（AS-81S：29,800円），PC-9800のプリンタ出力セントロニクスに接続し，モジュラーに変換する「アダプタ」（PAS-8N：8,900円），モジュラーをD-SUB25ピンに変換する「コネクタ」（SDM-PS：1,500円），バッファメモリを搭載した「プリンタネットワーク対応プリンタバッファ「TURBO Jシリーズ（JXM）」（256KB～16MB：59,800円～380,000円）があり，これらを組み合わせることにより，あらゆるオフィスにフレキシブルに対応し，多人数でのプリンタ共有を実現することができる．

沖電気工業，低価格の9600bpsモデムを発売

沖電気工業㈱（沖電気PCLINKインフォメーションセンタ：TEL 0120-296-007）は，最高通信速

ワンラインNEWS
メモレックス・テレックスは日本IBMのオフコン「AS/400」用のカートリッジテープ装置「MEMOREX TELEX 5420」を開発，販売を開始した．

NEWS & NEW PRODUCTS

度9600bps，同クラスでは業界最安値（98,000円）を実現した高速，普及型のパソコンモデム「PCLINK 296EX」を発売する．CCITT勧告V.32に準拠し，V.42bis，MNPクラス5の通信プロトコル機能を搭載している．コマンドは，Hayes ATコマンドによりモデムの設定，制御が可能で，設定内容は不揮発性メモリに記憶されるため，電源を切っても内容は消去されない．Macintosh用ケーブル付き，PC-9800シリーズ用ケーブル付きの2機種がある．出荷は2月上旬の予定．

マイクロソフト，Microsoft Worksをバージョンアップ

マイクロソフト㈱（インフォメーションセンター：TEL 03-3363-1201）は，統合ソフト「Microsoft Works（以下MS-Works）」をVer.2.5にバージョンアップした．「MS-Works」は，ワープロ，表計算，データベース，通信の4つの機能を統合，すべてのツールの操作性を統一し，各ツール間でのデータのやりとりも自在に行うことができる．「MS-Works Ver.2.5」では，矢印キーで自由に罫線を引くことができる罫線機能，いままでの文字種類に加え新たに縦倍角，4倍角，斜体などの文字種類を拡張，スペルチェッカーの装備，縦書き印刷のサポートなど，ワープロツールを中心に機能強化がなされた．価格は40,000円で，Ver.2.1ユーザーには15,000円でバージョンアップを行う．また，11月25日以降にVer.2.1を購入したユーザーには無償でバージョンアップを行う．

エルゴソフト，DOS/V対応日本語ワープロソフトを発売

㈱エルゴソフト（TEL 03-3589-4951）は，DOS/Vに対応した日本語ワープロソフト「EGLight Ver1.1」を発売した．共通ファイルフォーマットEDF形式をサポートし，フロッピーにより同社のMacintosh版EGWordとのファイル互換が可能，ネットワークを考慮したコンパクトサイズ，日本語入力フロントエンドプロセッサ「EGBridge」を標準装備している．価格は15,000円．ほかに企業向けパック84,000円（10本），367,500円（50本），3,000,000円（コピーフリー版）がある．また，今回のDOS/V対応版の出荷にともない，既存のIBM版の出荷を停止，IBM版ユーザーに対し，希望者には3,090円で交換する．

アルパソフト，Windows用ツール集，統合型アプリケーションを発売

アルパソフト㈱（TEL 03-3863-5976）は，Windows 3.0に対応したシステム構築・管理ツール集「Win-Tool Ver.1.0」を発売，また，統合型アプリケーション「Estella（エステラ）Ver.1.0」を4月に発売する．「Win-Tool」は，ドライブアイコンをクリックするだけで，複数ドライブのツリーウィンドウ表示が可能なファイルウィンドウの表示能力，複数のドライブを対象としたファイル検索が行え，検索したファイルをそのままタイプ・ダンプ・複写・削除することのできる強力なファイル操作機能，アプリケーションを登録することにより，ボタンをクリックするだけでアプリケーションを実行することのできるアクセサリーボタンのほか，MS-NETWORKSをサポートするNet-Toolを標準添付している．価格は9,800円．「Estella」は，カード型データベース，DTP機能などをもつ統合型アプリケーションで，カード型データベースでは，T型，カード型，リスト型，およびそれぞれを組み合わせた画面でのデータ編集，複数の項目に対して条件を指定した検索，辞書順ソートが行える．DTPでは，印刷イメージを画面上で確認できるWYSIWYGの実現により，より高度な編集が行える．また，表作成機能や簡易文章データベースの利用により一般的なビジネス文書も容易に作成することができる．価格は49,800円．

Programmer's Program

低価格なMASM互換アセンブラ

Light Macro Assembler Version 1.0

ハードに密着したプログラム，実行速度が重要なプログラム，コンパクトさが大きなウエイトを占めるプログラムなど，いまでもアセンブラが活躍する場面を数多く見ることができる．インラインアセンブラ機能が使えるコンパイラの登場もアセンブラの必要性を物語っている．ところが，コンパイラの低価格化によって高級言語の入門者が増加したのに対し，アセンブラはプロフェッショナル向けの感が強く，また勉強のために購入するには高価だった．「Light Macro Assembler Version 1.0(LASM)」は，こんな常識を覆す低価格のアセンブラである．

標準性に富んだ設計

LASMのおもな特徴として次のような点をあげることができる．

標準性

LASMは80286までのCPU，数値演算用コプロセッサ8087，80287の命令を生成することができる．

LASMの文法は，MS-DOS上のアセンブラの実質的な標準である「Microsoft Macro Assembler Version 5.1 (MASM 5.1)」の文法仕様とほぼ同じである．ただし，マクロ構文に一部対応していない．MS-DOSのプログラマーズリファレンスに掲載されているアセンブラソースや，Microsoft C Professional Development System (MS-C) やTurbo Cで生成されたアセンブラソースを，ほとんどの場合手直しすることなくLASMでアセンブルすることができる．

また，LASMはMS-DOSの標準形式のオブジェクトファイルを生成するので，MS-C，Turbo C，QuickC，QuickBASICなどの高級言語とのリンクが可能である．

高速／コンパクトなコード

LASMはマルチパス方式の採用により前方参照を完全に解決しているので，余分なNOPコードの生成やフェーズエラーによるアセンブルの失敗がない．また，オプションの指定でジャンプ命令の最適化が可能である．最適化時には，SHORTタイプでは届かないジャンプ命令を自動的に解決し，JMP命令と組み合わせたコードを出力する．

わかりやすさ

膨大になりがちなマニュアルは，約170ページと簡潔にまとめられている．200種類以上のエラー／警告メッセージを日本語で的確に出力する．ほかのエラーに起因する派生的なエラー／警告メッセージの出力を抑制することができる．マクロ中でエラーが発生したときは，そのマクロへの呼び出し経路を表示する．

プログラムの大きさ

LASMはメモリ容量の許すかぎりどんな大きなプログラムでもアセンブルが可能で，またディスク容量が許すかぎりどんな大きなマクロでも使用できる．アセンブル速度はCPUに80286を搭載した機種で1分当たり約10,000～15,000行である．

日本語への対応

ラベルやマクロの名前に全角文字や半角カナを使用することができる．15種類以上のアセンブルオプションにより，アセンブル方法をさまざまなに制御できる．アセンブルリストが作成可能で，そのリストを再アセンブルすることもできる．リスト中のアセンブラコードは，生成コードの統一的な表記になっており，生成されたコードの確認に役立つ．またセグメント，ラベル，マクロについての要約表を日本語で作成し，簡略化セグメントディレクティブが使用可能．

リンカ

LASMには，ユーティリティなしでCOM形式の実行ファイルを直接出力することができるリンカ「LIL」が付属している．また，MS-DOSに付属するリンカ「LINK」を使うこともできる．

ワンラインNEWS
米オーサウェアは日本でのソフト事業を強化する．マッキントッシュやUNIX用マルチメディア関連ソフトを日本での独占販売権をもつアスキーを通して販売する．

デバッガ

LASMにはソースレベルデバッガ「LID」が付属している．ただし，デバッグできるプログラムはLILでリンクされた実行ファイルのみである．デバッガが使用する割り込みベクタは標準ではINT3が設定されているが，INT3を利用するプログラムをデバッグするときのためにオプションで任意のベクタを指定することもできる．また，実行経路の追跡表示のみを行うモードがある．もちろん，MS-DOSに付属の「SYMDEB」によるデバッグも可能である．

アセンブラ入門に適した手軽さ

最近の処理系製品に見られるような統合環境やツール類をつけたいたれりつくせりの製品ではないが，その軽さが入門者にはメリットになる面もある．

パッケージを開けると

パッケージには，3.5インチまたは5インチのディスク1枚と約170ページのマニュアルが1冊入っている．マニュアルには，インストール，アセンブラやリンカそしてデバッガの簡単な利用方法とオプションの説明があり，あとはアセンブラの文法の解説だけである．DOSのプログラマーズリファレンスに掲載されるサンプルマクロや，アセンブラソースの公開されているフリーソフトウェアなど，参考となるソースは数多くあるし，市販のアセンブラ入門書なども多数出回っている．これらを活用すれば，分厚く開くのもおっくうなマニュアルを使うより的確に必要な情報が見つけられるかもしれない．

いざアセンブル

試しにTurbo Assembler用に書いた自作のアセンブラソースを2，3アセンブルしてみたが，一部の疑似命令に手直しが必要だったことを除けば，どれも実行形式ファイルを生成することができた．もちろん，生成された実行形式ファイルはどれも問題なく動作した．ただしこれらのプログラムは短く，基本的な命令しか使っていないため，そのまま大きなソースファイルには当てはまらない可能性もある．

日本語のエラーメッセージ

考えてみれば，コンパイラやアセンブラが出力するメッセージは，なぜか英語である．メッセージの意味を調べるために分厚いマニュアルのエラーリファレンスを引くのに苦労している人も少なくないだろう．その点LASMのメッセージはすべて日本語化されているため，エラーメッセージを見ればどんなエラーかすぐに検討がつく．入門者には嬉しい配慮である．

8ビットのパソコンが全盛だったころは，「パソコンが使える」=「プログラムが書ける」だった．ところが，16ビットのパソコンは当初高価だったため趣味よりビジネス優先で進化してきた．「パソコンが使える」=「ワープロや表計算が操作できる」になった．しかし，「パソコンが使える」=「プログラムが書ける」をめざす人は少なくないのである．しかも，その人達は8ビットパソコン全盛時代のように一部のマニアではない．C言語やPascalでプログラムが書けてあたりまえの現在，これからの「1歩上を行く見栄パソコンユーザー」にはアセンブラは必須ではないだろうか．

■ **Light Macro Assembler Version 1.0**

販売：（有）伊澄システム製作所（TEL 03-3488-5243）／価格：9,800円／適応機種：日本電気PC-9801シリーズ，エプソンPC-286，386シリーズ，富士通FM-Towns，FMRシリーズ，東芝J-3100シリーズ，各社AX／動作環境：MS-DOS Ver.2.11以降／必要メモリ：384Kバイト／供給メディア：5インチ2HDまたは3.5インチ2HD

オンラインソフト採集帳 14

愛の潜水艦シューティングゲーム Super Depth

ASCII-NET PCSのpoolやNIFTYのライブラリには，数多くのオンラインソフトが登録されている．これら登録されたオンラインソフトの中で，ダウンロード数が多いジャンルのひとつにゲームがある．「高価なパソコンでゲームなんか」と抵抗のある方もいると思うが，ゲームは話題のグラフィカルユーザーインターフェースや高速スクロールなどを駆使し，パソコンの性能を極限まで引き出している．大げさにいえば最先端の高度なプログラミングテクニックは，ゲームプログラミングからフィードバックされているのである．

今回紹介する「SuperDepth」は10年ほど前ゲームセンターで流行した「潜水艦ゲーム」を元に大幅なリメークをかけたゲームである．

準備

SuperDepthは，日本電気のPC-9801シリーズのうちVM以降の機種に対応している．ただし，VM2やVF2などはオプションの16色グラフィック拡張ボードが増設されている必要がある．また，ドキュメントにはCPUにV30を使った機種では処理が重くなってしまう場合があるので，80286以上のCPUを搭載した機種が推奨されている．もちろんFM音源ボードのジョイスティックにも対応している．

ソフトウェアはOSにMS-DOS Ver3.3を用意する必要がある．また，VSYNC割り込みやタイマ割り込みを使用しているので，これを使用する常駐終了型プログラムとは相性の悪い場合がある．起動するにはかなりの空きメモリを必要とし，仮に起動できたとしてもゲーム中にメモリ不足でハングアップするなど不都合が発生するので，あらかじめ日本語入力FEPなどや常駐終了型プログラムを外し，十分な空きメモリを確保する必要がある．

起動方法

実行するにはまず，DEPTH100.LZHから復元されるすべてのファイルを同じディレクトリにおく必要がある．パスが通っていればどのディレクトリからでも実行可能だが，これらのファイルは必ず同じディレクトリにおかなければならない．ただしハイスコアを記録するファイルであるDEPTH.SCRがない場合には，あとで自動的に生成される．

「SuperDepth」の起動はMS-DOSのコマンドラインから

```
A> DEPTH
```

と入力する．起動時SuperDepthは2つの起動オプションを指定することができる．-Xオプションは，左右のトリガを逆にする指定である．これはジョイスティックなどを使ってゲームをする場合，トリガの位置が逆のほうが使いやすい場合などに設定する．-Gオプションは，CDGのクロックを強制的に2.5MHzにする指定である．PC-9801 VMなどで画面がおかしくなったときはこのオプションを設定することで回避できる．

操作方法

起動すると，まず「Bio_100%」のタイトルが表示され，次にデモンストレーション画面になる．ここでテンキーの[2]，[8]やカーソルキーの[↑]，[↓]で「ゲームのスタート」，「ハイスコア画面」，「ゲームの終了」を選択し，[Z]，[X]または[スペース]でカーソル位置の実行となる．デモンストレーション画面やハイスコア画面で[Q]または[ESC]を押すと，SuperDepthを終了しMS-DOSのコマンドラインに戻る．またゲーム中は[ESC]または[Q]を押すことによって，随時一時停止することができる．このとき[Q]を押せばゲームは強制終了となり，MS-DOSのコマンドラインに戻る．そのほかのキーを押せば，一時停止の解除となる．

ゲームは海，空，宇宙，ボスの各ステージが用意されていて，全12ステージをクリアするとエンディングをみることができる（とドキュメントにあるが，下手なためまだ未確認(笑)）．キー操作はテンキーの[2][4][6][8]やカーソルキーなどで「戦艦ヤマト」を移動（海と空のステージでは「戦艦ヤマト」は縦方向の移動はできない）し，[Z][X]で弾を発射する．スクリーン内にある敵機を破壊するごとに得点が加算されていく．レーダー上で白く点滅する敵機を破壊するとアイテムを出す．このアイテムは，

- 青：移動速度アップ
- 赤：連射数アップ
- 緑：弾のスピードアップ
- 紫：3方向爆撃（海面は波状爆撃）
- 水：上記のすべて

ワンラインNEWS
群馬日本電気はパソコンの組立・検査の自動化ラインを導入し，稼働を開始した．これにより製造工程の短縮や品質の向上によるコストダウンが可能になった．

黄：画面内の敵を壊滅
白：１機増える

の７種類が用意されている．画面下部のレーダーを参考にして攻撃すれば高得点が狙えるだろう．敵機の放った弾や敵機に当たると「戦艦ヤマト」は減っていき，３機あった「戦艦ヤマト」がすべてなくなるとゲーム終了となる．迫りくる敵を撃破し，目指すは華麗なるエンディングである．

「Bio_100%」ブランドは，新感覚アクションゲーム「蟹味噌」や2Dドライブゲーム「か～にグランプリ」，そして3Dドライブゲーム「ろりろりろ～りんぐ」など，多くのオンラインゲームを輩出している．また「Studio☆Femy」は「Bio_100%」ブランドのゲームのほかに，「joylib」や「grlib」などのライブラリ，そして高視野ページャー「wlook」などのツールもオンラインで数多く発表しているBBSである．ただし，現在「Studio☆Femy」はドキュメントにかかれている電話番号でアクセスすることはできないので注意が必要である．

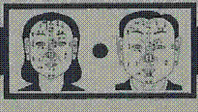
■SuperDepth Ver.1.00

名称：SuperDepth／分類：シューティングゲーム／登録名：DEPTH100.LZH／作者：森栄樹（alty,ASCII-NET PCS pcs36125），二木康夫（tacox,ASCII-NET PCS pcs35738）およびBio_100%／動作環境：PC-9801シリーズ（VM以降）／登録月日：1991年８月11日／復元方法：LHA E DEPTH100／入手場所：ASCII-NET PCS（pool msdos #4241）

ファイル構成

Listing of archive : DEPTH100.LZH

Name	Original	Packed	Ratio	Date	Time	Attr	Type	CRC
DEPTH.BGM	18127	3559	19.6%	91-08-10	01:00:00	a--w	-lh5-	29D3
DEPTH.BOS	28704	13590	47.3%	91-08-10	01:00:00	a--w	-lh5-	A425
DEPTH.C08	224	120	53.6%	91-08-10	01:00:00	a--w	-lh5-	CA5E
DEPTH.C16	14368	3495	24.3%	91-08-10	01:00:00	a--w	-lh5-	6621
DEPTH.C32	32800	7205	22.0%	91-08-10	01:00:00	a--w	-lh5-	5F5B
DEPTH.DOC	6800	3622	53.3%	91-08-10	01:00:00	a--w	-lh5-	6FCA
DEPTH.EFS	1737	339	19.5%	91-08-10	01:00:00	a--w	-lh5-	8B57
DEPTH.EXE	70731	37243	52.7%	91-08-10	01:00:00	a--w	-lh5-	72DF
DEPTH.FNT	8224	1841	22.4%	91-08-10	01:00:00	a--w	-lh5-	830F
DEPTH.SCR	291	138	47.4%	91-08-10	01:00:00	a--w	-lh5-	3EC2
DEPTHBGM.DOC	7949	3759	47.3%	91-08-10	01:00:00	a--w	-lh5-	9C8C
11 files	189955	74911	39.4%	91-08-11	00:23:44			

HEALTHY?  UNHEALTHY?

14 電磁波の謎②

CRTディスプレイは「人体に有害なのか」というテーマのつづきである．

そもそもCRTディスプレイと家庭用テレビに本質的な違いがあるのだろうか．CRTはCathode-Ray Tubeの略で陰極線管と訳されている．これは，ブラウン管と同じものである．真空の管内で電子を発射し，蛍光物質を発光させることによって，像を結ぶという原理だ．普通のテレビも同様の仕組みで働いている．広範に普及してから30年にも及ぶテレビで問題がなければCRTについてもあまり神経質になることはない．

幸いなことに，テレビが普及しはじめたころに心配された「電磁波による障害」は発生していない．少なくとも社会問題にはなっていない．テレビとパソコンのディスプレイとでは，目からの距離が異なるという意見もあるかもしれないが，モニターと向き合って仕事をすることの多いテレビ局でも障害は報告されていないらしい．

しかし，パソコンが普及する前に，大型機の端末としてディスプレイが登場したころは，さまざまなOA病が報告されている．「目が極度に疲れる」「肩凝りがひどい」などのOA病の症状は，どこまでがディスプレイやキーボードによるものなのか判断しにくい．とくに，OA化の当初は，新しいシステムに対する心理的障壁が高く，それが身体症状としてでたことも想像に難くない．OA化をすすめることで結果的に労働過重になっていたのである．この点については，適度な休憩をとるということで対策が行われている．

では，CRTはまったく安全で，ディスプレイにつける防護フィルターは役に立たないのかといったら，そうでもない．フィルターは画面のまぶしさや反射を防いでくれるからだ．輝度の調整不良や画面のちらつきは，目の疲れに直結する．このフリッカ（ちらつき）とグレア（まぶしさ）の問題については次号でお話しする．

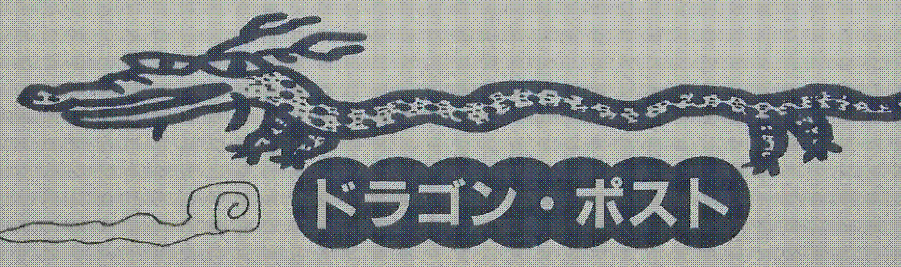


Illustration
Nobuki Fukuhara

この欄は，読者からのお便りを中心に構成しています．本誌に対するご意見・ご感想などをお待ちしております．鬼より恐い読者からの声ということで，「ドラゴン・ポスト」です．本誌についているアンケートはがきにご意見の欄を設けてありますから，そこに書いてください．ペンネームでも歓迎です．ただし，はがきには本名と住所をしっかり書いてくださいね．掲載誌をお届けします．

★どのマシンがいいんでしょう？

自分用にPCがほしい（J31はカミさんにとられっぱなし）と思っているのだが，ここ１～２年で98とPC/AT＆アップルの勢力に変化がおきそうで，98に手が出せずにいる．RISC PCにも目がはなせないし，WIN3 vs. OS/2のどちらを取ったら良いのか，UNIXに行くのか，SUN OSに行くのか……．私がPC-8001を買った時，MZにしようか，タンディにしようかとまよったあげくPC-8001にしたけれど，今回はまったくもってわからん！　もう２～３年じっくり待とうとも思うけど，早く手にしたい気持ちもあって，ウーム．何がいいと思いますか？（柴田和広）

――やはりなんといっても，MS-DOSもOS/2もUNIXも動作するIBM PC/ATコンパチの486マシンで決りでしょう！

★目指せベンチャー企業

誌上などでは活発なDOS/Vですが，まだまだですね．町中のパソコンショップでは，でも，私は機は熟しているので，早く市場に広まってくれるとうれしいです．America，Taiwan，Koriaなどからの個人輸入を手伝ってくれるShopがでてきてもよいと思うんだけど，どうでしょう？　そんなベンチャー企業ないかなァ？　資金とKnow howがあれば私もやりたい（？）（目指せ企業家の狂夢）

――直接個人輸入の代行をやってくれるところは知りませんが，秋葉原などのショップにはすでに安いIBM PC/AT互換機が出回っていますよ．

★ダイヤルQ2にすれば……

宮沢りえの写真集が話題になっているが，ビデオカメラで画像データとして取り込み，Net上にアップして，ダイヤルQ2でアクセスしてロードすれば，あんな大騒ぎにならないのになあ……．（JUN）

――でも，そんなことをしたらダイヤルQ2の回線がパンクしてしまうのではないでしょうか？　編集部にもあの写真集を買ってきた女性が１人いますよ．

★難しいゾ！　ランダムスクリーン

今月のランダムスクリーンは，はっきりいってむずかしい．記事の写真だけではまったくわからず，FDを展開してやっとわかりました．FDが使える人には簡単ですが，98，互換機をもっていない人には今回の問題はとけないと思います．最初，見た時は季節がらかずの子かと思いました．また，この手の問題は画面を見て，正解はわかりますが，実際に正しい配列にするのはとんでもなく大変です．場所がわかるようにすると簡単になってしまうので，接続状態が正しい時，その方向の枠が消える機能があればと思いますが，いかがでしょうか．（岩崎美之）

――う～ん，あのソフトを作った本人にも元通りには組み立てられない状態ですからねぇ．バージョンアップ……するのかなぁ？

★誠に良い趣味

毎月パソコン誌に付録で付いてくる各種のプログラムソフトの整理の為に，会社を休暇で一日休んでいるが，最近は一日では整理し切れない．８冊も買って，それぞれに全て付録付なので，今月号（12月号）では８枚を全て展開したが，どんどんFDが溜まっていく．中にはDOCファイルすら読めないまま，次月号ということになるものもある．それでも楽しい!!　誠に良い趣味だと思っている．（長谷山隆嗣）

――そんなことをするために会社を休んでしまっていいのでしょうか？　編集部でそんなことをしたら，フクロにされてしまいますけど．

★この１年いろいろありました

プログラマーズ・ページも本棚に12冊ならびました．１年ははやいのかおそいのか，人それぞれですけど，私は286から386へ，640KBから3.6MBへ，フロッピーからハードディスクへ，リッチからプアーへ，いろいろありました．（PONYO2）

――１年は，ほんとうに早いですね．ついこの間創刊したばかりだと思っていたのに……．果たして，もう12冊並ぶのでしょうか？

★悩みはつきない……

親元に帰るたび，禁断症状になやまされる．しかし学生

である以上，２台目のパソコンを買う金はないし，机の上にパソコンがないとやることがないというか，おちつかないというか．うーん，来年就職したら286VJにするか，H-98にするか(つまり将来性か経済性か)悩みはつきない．(乾聖司)
——IBM PC/AT互換機というのはどうですか？　あるいはMacintoshとか？

★表紙が最高!!
本の“造り”(内容ではない!!)がすごくいい．表紙が最高，使っている紙も他誌と較べ上質!!　内容はその月の特集による．10月号はとてもよかった．全体的にはもう少しほりさげた特集になったら良いと思う．(有地宏史)
——他紙に比べれば，ずいぶん掘り下げた内容ではないかと思っているのですが，無限に紙面があるわけではありませんから，どうしても限界はありますね．

★気になるDOS5
NECよりMS-DOS5が出た！　教育現場にもパソコンをと，業者(代理店)が説明に来ていた．うちの管内に入るというので，ずいぶん熱心だった．……が，私はMS-DOS5の方が気になって仕方がないので，その事ばかりきいた．しかし，説明の中で「タスクスイッチャーは一時ハードディスクへ退避して……」．こんなんで，だいじょうぶかな．でも親切でした．
PS.FEPはどうなるんだろう……．(TOKUTOKU)
——編集部にDOS5が来た日には，あちらこちらから悲鳴が聞こえました．きっとあれは，喜びの悲鳴だったんでしょう？！

★BASICの特集がよかった！
いままで，書店で表紙を見るだけぐらいだったのですが，今回の特集「やっぱり僕らはBASIC！」を見て感心しました．私もBASIC，FORTRANからはじまって，Turbo PASCAL，C，C＋＋と，個人的に学んできたのですが，個人的な好みから言うと，BASICのあいまいさは好きです．PASCALのガンコさ，CやC＋＋の要領の良さなど，それぞれの言語には良い点がたくさんあります．それでもBASICは英語のようなとっつきの良さと，ふところの広さがあるような気がしますね．Turbo C＋＋とTrueBASICともに楽しんでいきたいと思う自信がつきました．アリガトー．(川田幸男)
——まだまだBASICを使用しているユーザーは多いようですね．QuickBASICの連載も始まりましたので，こちらもよろしく．

★Cだけではない！
ふと，前号までの「ドラゴン・ポスト」を読むと，C言語の氾濫に文句を言う人が多いことに気付きました．私の考えでは，踊っているのは一般ユーザーが多数だと思います．10月号のコメントは正しいと思います．現在，COBOLでプログラムし，一部アセンブラで色をつけています．C言語の仕事が多いとは思いません．(生形洋一)
——日本国内においても，世界においても，いちばん人数の多いのはCOBOLのプログラマだといわれています．言語の人気と実際のユーザー数とは，かならずしも一致しないということでしょう．

★フロッピーバージョンが欲しい！
最近はハードディスク専用の言語処理系が増えてきましたけど，いくら安くなったとはいえ，ハードディスクは我々学生にとってはまだまだ高嶺の花．幸い，私はフロッピードライブを４つもっているので，MS-C ver6などをフロッピーに展開して使用していますけど，メーカーさんもフロッピーバージョンとか出して，フロッピーONLYユーザーに対処してもらえないものでしょうか．
P.S.来年からコンピュータ専門学校の教師になります．(MOAI)
——ハードディスクは必需品です．フロッピーONLYのマシンを発売するメーカーに問題があるのです．

★ついてけない……
Pascalでプログラムを作成しています．Ver.3.0をのんびり使っていたら，4.0でユニットが使えるようになって，5.0になって，5.5でOOPになり，6.0にバージョンupしたら，今度はTurbo Pascal Windows．エーン．ついていけないよー(野瀬幸雄)
——進化した部分はほっておいて，基本的な部分だけを使ってプログラミングしていれば大丈夫ですよ．マイペース，マイペース．



付録ディスクのご案内

このフロッピーディスクに含まれる実行ファイル，ソースリストおよびテキストファイルは「月刊Programmer's Page」1992年２月号に掲載されたものです．

付録のフロッピーディスクは，5"2HD（８セクタ／トラックMS-DOS）でフォーマットされています．3.5"や2DDなどへのコンバートサービスは行っておりません．なお，オンラインサービスSE-NET（03-5276-1297）でも，同じ内容のファイルを用意しています．

フロッピーディスクの内容は次の通りです．本誌およびディスクに含まれるドキュメントファイルをよく読んでからご利用ください．

■ディスクのおもな内容

●付録ディスクについての説明

ディレクトリ名：ルート

収録ファイル名：

README.BAT（ディスク内容の説明を読むためのバッチファイル，PC-9801専用）

README.DOC（ディスク内容の説明，MS-DOSテキストファイル）

ONLINE.BAT（オンラインソフトの説明を読むためのバッチファイル，PC-9801専用）

ONLINE.DOC（収録されているオンラインソフトの簡単な説明）

V.EXE(ミニページャー，PC-9801専用)

BOOK.TXT（翔泳社書籍目録）

●特集「通信ソフトの作り方」

②「基礎編：通信ライブラリを作ろう」(p.8)

ディレクトリ名：TOKU

収録ファイル名：TOKU2.LZH

③「実践編：超お手軽ターミナル」(p.31)

ディレクトリ名：TOKU

収録ファイル名：TOKU3.LZH

●プログラム「遊」工房(p.72)

「colax／落下型パズル」ゲーム

ディレクトリ名：PLAY

収録ファイル名：PLAY.LZH

●イカすBASICプログラム入門(p.83)

冬休みの構造化制御構文編

ディレクトリ名：BASIC

収録ファイル名：QBASIC.LZH．

●実践C塾(p.92)

ソート（並べ替え）・その２

ディレクトリ名：JUKU

収録ファイル名：JUKU.LZH

●98活用アセンブラ技法(p.102)

SCSIデバイスドライバ／SCSIテープドライバの作成

ディレクトリ名：98ASM

収録ファイル名：98ASM.LZH

●綴じ込み付録

ディレクトリ名：DATA

収録ファイル名：ATCOM.TXT（MS-DOSテキストファイル）

●オンラインソフト採集帳(p.136)

本誌今月号のオンラインソフト採集帳で紹介した潜水艦シューティングゲーム「Super Depth」と高圧縮復元書庫管理ソフト「LHA」が収録されています．拡張子が「.LZH」のファイルの復元方法についてはLHAに含まれるドキュメントを，LHAの簡単な使用方法についてはONLINE.DOCを参照してください．

ディレクトリ名：ONLINE

収録ファイル名：

DEPTH100.LZH（Super Depth v.1.00）

LHA213.EXE（LHA Ver.2.13，MS-DOS汎用）

●クイズ・ランダムスクリーン(p.142)

クイズ解読ツール「PUZLE」

本誌掲載のクイズ・ランダムスクリーンの画像をディスプレイ上に再現し，マウスを使って並べ替えることにより，解答を得ることができるユーティリティです．PC-9801シリーズ専用です．詳しい使い方はPUZLE.DOCを参照してください．

ディレクトリ名：OMAKE

収録ファイル名：

PUZLE.EXE(解読ツール，PC-9801専用)

PUZLE.DOC（PUZLEの説明）

CONFIG.SYS（PUZLE用CONFIG.SYS）

AUTOEXEC.BAT（PUZLE用AUTOEXEC.BAT）

RNDSCREN.LZH（画像データ）

SARU.LZH（１月号の解答データ）

SARU.BMP（Windows 3.0の壁紙）

※使用上の注意

本誌に掲載されたプログラムおよびデータを利用するには，MS-DOSの基礎知識が必要です．このディスクには，MS-DOSのシステムは入っていません．別途，MS-DOSが必要となります．また，ソースファイルのコンパイルなど，別途コンパイラやユーティリティが必要な場合があります．

■圧縮ファイルについて

ディスクスペースを有効に使うために，多くのファイルは圧縮した上で収録しています．「.LZH」という拡張子のついたファイルは，高圧縮書庫管理ユーティリティ「LHA」によって圧縮されたファイルです．これらのファイルを利用するには，まずLHAで復元する必要があります．LHAの利用方法については，ONLINE.DOCをご覧ください．

バーチャルスペースガイド

C平　マルチメディアマガジンが出たんだって．ちょっと前の話になるけど，マルチメディア'91で見本版のレーザーディスクをもらってきちゃった．「MEGA-MIX Magazine」っていうんだ．

P子　へー，レーザーディスクでね．たしかに，媒体としてはプレーヤーが普及しているから可能性はあるんだろうけど，あれはビデオディスクなんだから，いまさらマルチメディアでレーザー？という気もするけど……．

C平　中身は動画と静止画を組み合わせて構成しているんだ．「SIGGRAPH'91バーチャルウオーキング」というのがあるんだけど，静止画のコマ送りでマルチメディア風の画面にしているというわけ．

P子　何か変じゃない．

C平　パソコンでレーザーディスクをコントロールするというのは前からあって，シリアルコネクタが付いているレーザープレイヤーなら可能なんだ．マックでコントロールする美術館シリーズも発売されているし，ただし，そこまでやるとすると販売ターゲットがすごく少なくなってしまう．つまり，このマルチメディアはインタラクティブじゃないから，従来のレーザーディスクにマルチメディアの情報を入れて，見せかけもマルチメディアっぽくしているというものだね．

P子　つまり，マガジンという電子出版的なものというより，ビデオソフトの分野に入るものなのね．

C平　でも，こういう試みが行われるということは，おもしろい．開発技術の話ばかりでは，なかなか進まないのだから．ただ，6号分で5万円以上という価格では，手を出しにくいけど……

■MEGA-MIX Magazine プレミアム版

MEGA-MIX Communications
仕様：20㎝レーザーディスク

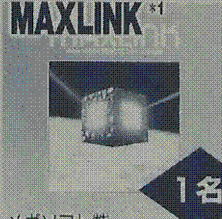
4000枚以上の静止画で「SIGGRAPH'91」をインタラクティブに仮想体験できる「バーチャルウォーキング」，デジタルサウンド付きの「CGシアター」，コンピュータ業界のキーマンへのインタビューなどの「VIDEO REPORT」，そして「MEGA-MIX NEWS」により構成される．本誌は，11月に創刊され以後隔月刊．定期購読のみでマガジン版が6号分54,000円，マガジン版にエクストラ版を加えたものが6号分で96,000円．



読者プレゼント

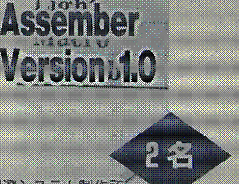
応募は本誌に添付のアンケートはがきをご利用ください．（1992年2月17日消印有効）．なお，応募者多数の場合は，抽選になります．当選者発表は発送をもって代えさせていただきます．

1 MAXLINK *1



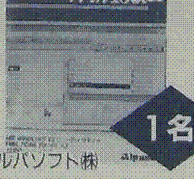
1名
提供：メガソフト(株)

2 Light Macro Assember Version 1.0



2名
提供：(有)伊澄システム製作所

3 Win-Tool Ver.1.0 (Windows 3.0対応)



1名
提供：アルパソフト(株)

4 パソコンのお医者さん



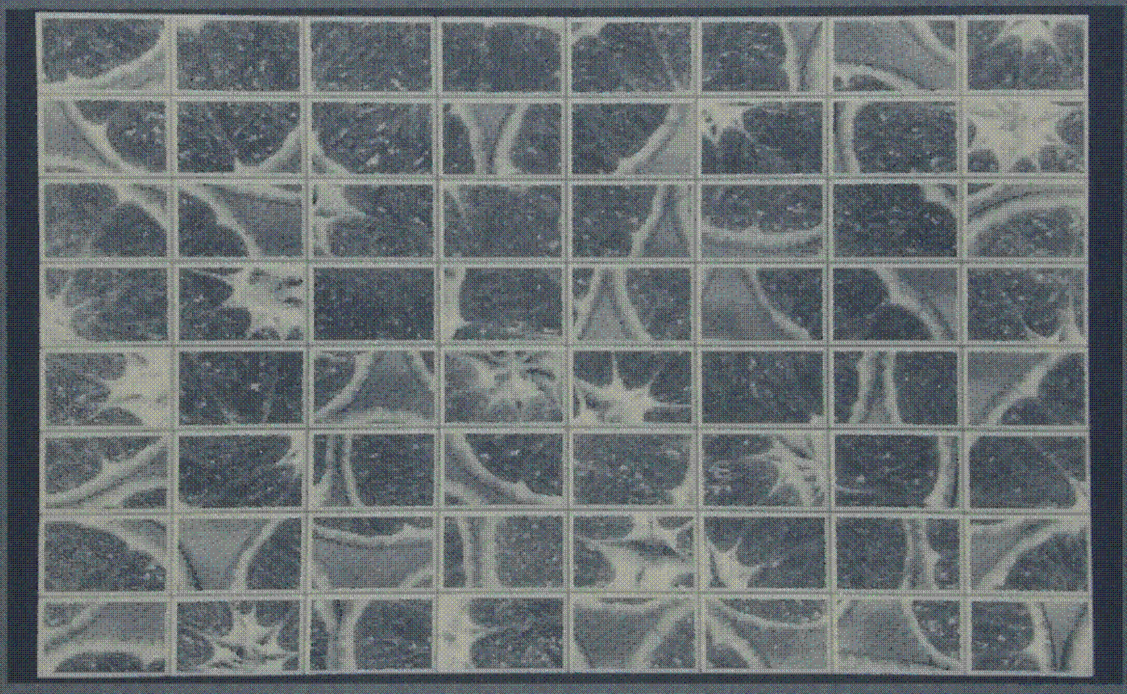
3名
提供：翔泳社・PC-PAGEシリーズ新刊

*1：IBM PC/AT，J-3100，他互換機対応

クイズ・ランダムスクリーン

ある１枚の画像を64分割し，ランダムに並べ替えてあります．
さて，もとの画像は何を写したものでしょうか．

ヒント：冬はやっぱり……



正解者の中から30名様に本誌表紙イラスト（日暮修一画）入り月刊プログラマーズ・ページ特製オリジナルマグカップをさしあげます．

応募方法：本誌に添付のアンケートはがきをご利用ください．

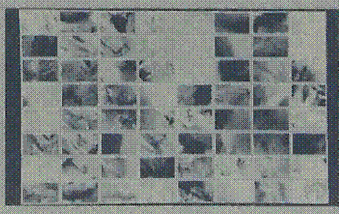
締め切り：1992年２月17日（消印有効）

なお，当選者の発表は発送をもって代えさせていただきます．

＊画像データは，付録ディスクやBBSでもサービスしています．

１月号の問題と正解

【問題】



【正解：猿】



SE-NETからのお知らせ

SE-NETでは開設以来，オンラインサービスを行ってきましたが，回線混雑のため非常に接続しにくい等の問題が出てまいりましたので，GUESTによるダウンロードは中止させていただきます．今後は読者のみなさまと編集部とのコミュニケーションの場としてご利用ください．

●BBSへの接続

SE-NET　03-5276-1297

通信速度	300/1200/2400bps
接続プロトコル	CCITT V.22bis/V.22/V.21
エラー訂正プロトコル	CCITT V.42，MNPクラス５
漢字コード	シフトJIS

月刊プログラマーズ・ページ バックナンバーのご案内

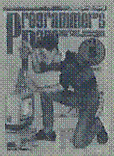
Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

■定価・各980円

（1991年1月号「特集・プログラマの仕事場」・2月号「特集・Windowsの時代である!」・3月号「特集・この言語を使いなさい!!」・4月号「特集・C読本」は品切れです）

1991年5月号 5インチFD付き

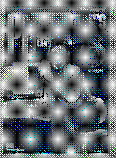
特集・基礎力としてのアセンブラ



プリミティブなプログラミング言語でありながら，それ故に無視することができないアセンブラ，コンピュータを理解し，その能力をひきだすためにもう一度アセンブラをみなおす．

1991年6月号 5インチFD付き

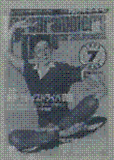
特集・研究「プログラミング統合環境」



今流行のエディタ，コンパイラ，リンカ，デバッガを統合した開発環境について，その歴史的系譜から最新のシステムまで徹底的に検証する．

1991年7月号 5インチFD付き

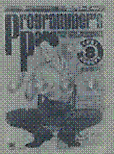
特集・最新「デバイスドライバ」研究



パソコンの拡張性を支えるソフトウェアインターフェースであるデバイスドライバ作成の技法と特徴をMS-DOS，Windows，OS/2の各環境ごとに解説する．

1991年8月号 5インチFD付き

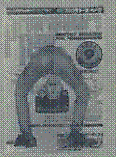
特集・はじめての オブジェクト指向プログラミング



"オブジェクト指向"の歴史的な発展の過程から基礎概念，分析・設計の方法までを網羅的に解説する．

1991年9月号 5インチFD付き

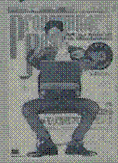
特集・グラフィックス プログラミングの世界



基礎からフレームバッファの活用，レイトレーシング，Windows，IBM PCのグラフィックスなど，その技法を徹底的に解説する．

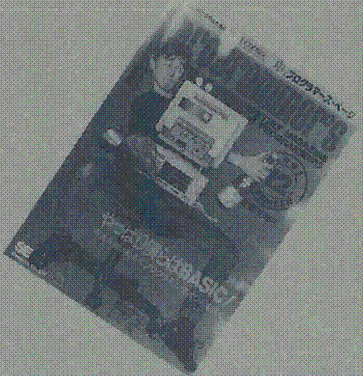
1991年10月号 5インチFD付き

特集・286/386 メモリ活用プログラミング



バンク式メモリ，EMS，XMSからプロテクトモードまで，640KBの壁を破るための数多くのプログラミングテクニックを解説する．







1991年11月号 5インチFD付き

特集・Windowsプログラムの作り方

正統派MS-C＋SDKをはじめ，Visual-Basic，BC＋＋，TPWなどの新しい開発環境，ツールによるWindowsプログラムの作り方を探る．

1991年12月号 5インチFD付き

特集・やっぱり僕らはBASIC/

構造化プログラミング，マルチランゲージに対応した強力なプログラミング言語として生まれ変わった現在の進んだBASIC，MS-BASIC 7.1，TrueBASICからVisualBasicまでを紹介する．

1992年1月号 5インチFD付き

特集・プログラミングツール大集合

プログラム開発，メンテナンスに欠くことのできないプログラミングツール，その利用法から，使いこなしのコツまでを紹介する．

「月刊プログラマーズ・ページ」のバックナンバーは，ご注文いただければ，全国どこの書店でもお求めになれます（書店に届くまで1か月ほどかかります）．小社に直接ご注文なさるときは，何月号かを明記の上，現金書留にて雑誌価格＋送料を右記までお送りください．送料は1部81円です（ただし7月号は1部86円）．2冊以上の場合は，号数・冊数により送料が異なりますので，小社までお問い合わせください．

〒102 東京都千代田区平河町
2-4-14 平河町KSビル4階
（株）翔泳社
マーケティング部PP係宛
TEL 03-3263-0447

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

月刊プログラマーズ・ページ3月号
(2月18日発売)

付録：5″ 2HDフロッピーディスク付き!!

特集
Cじゃなくても，いいじゃない

なぜか最近，Cばかりがもてはやされている．しかし，開発言語はCだけではない．C言語の問題点を明らかにし，C以外のプログラミング言語の能力と性能を実際のプログラミング例を示しながら解説する．

月刊プログラマーズ・ページ　1992年2月号
1992年2月1日発行

発行所　株式会社翔泳社
東京都千代田区平河町2-4-14平河町KSビル4F・5F
TEL 03-3263-0447　〒102
発行人　長廻健太郎
編集長　清水　隆
副編集長　鈴木嘉平
編集スタッフ　杉野貴美子＋上田純美礼＋芳山充弘
表紙デザイン　SE（翔泳社デザイン研究所）
広告スタッフ　相澤正夫＋澤本良知＋種村嘉彦
協力　根本裕三郎＋(株)ドモン・マインズ＋佐藤裕美＋小林弘人(図版デザイン)
写植　株式会社東巧社＋株式会社トーヨー・プリント社
印刷　凸版印刷株式会社
定価　980円（本体951円，税29円）

EDITOR'S PAGE

●コンピュータ業界のキーワードとしてマルチメディアが登場して久しいが，日本ではまだ具体化していない．すでに海外では，MPC（マルチメディアパソコン）への動きが始まっている．MPCでは，音源ボードが必須となる．PC/AT用の音源ボードとして有名なサウンドブラスタープロを軸として，CD-ROMドライブ，さらにWindowsのマルチメディアエクステンション版，Microsoft Bookshelf for Windowsなど5枚のCD-ROMソフトがセットされて販売されている．米国では850ドル，日本でも15万円程度で買うことができる．この価格とセットの内容にMPC時代の到来を感じることができる．海の向こうの「標準」の上でのことだが……．(TA)

●ついこの間までフロッピーベースの98があたりまえだと感じていたのに，今では486のPC/ATコンパチ機でメモリは目いっぱい，ハードディスクも300MB以上あるのが当然という感覚になっている．やはりDOS/Vの登場とともに増え始めた安価なIBM PC/ATコンパチ機によるところが大きいようだ．プログラミングの先端に位置する人々の目はすでにDOS/Vとその向こうに広がる膨大なIBM PC/AT互換機のソフトウェア市場に注がれているようだし，私もそろそろ486マシンの購入を真剣に検討したほうがよさそうだ．(嘉平)

●本誌の目次や扉で使用している写真は，特集のレイアウトや内容に合わせて，フォトライブラリで探しています．探すものが具体的に決まっている場合はいいのですが，イメージしか決まっていない場合は結構たいへん．キーワード別に箱が分かれてはいるのですが，たとえば12月号のBASICの特集の時は，「住居(海外)」にはじまり「フランス」「ドイツ」と数十箱を広げ，壁や石畳の写真を探し求めて数百枚，ほとんど世界一周の旅でした．きれいな写真が多いので，ふだんは楽しめる仕事ですが，今月号の写真（蜘蛛の巣）探しはつらかった．「昆虫」の箱の中には，超どアップのかまきりや蟻がウヨウヨ……．鳥肌を立てながらようやく見つけてきました．(ま)

●PC-9800シリーズ用のMS-DOS 5.0を求めて秋葉原のパソコンショップをさまよい歩きました．例によって，初期の出荷数は少なく，どの店も「売り切れです」「ありません」とつれない返事．みんなが欲しいと思っているのに，どうしてユーミンのCD並みに「ン十万本一挙出荷！」とはいかないのでしょう．日本電気さんは，ユーザーは皆おとなしくバージョンアップ通知を待つべきだと考えているのかしらん．あげくに新型のデータディスクマンを予約し，手ぶらで帰った私の1日は何だったのでしょう!?
(kos)

●V.32モデムの価格が国産機種で無理すれば手の届くところまで下がってきた．しかし，海外ではUS Robotics社のCOURIER HST Dual standardがV.32ter対応になるという話を聞く．我がモデムは未だにOLD HST……．米PC MAGAZINEでV.32モデムが$399.55という広告を見て心揺れるが，財布はまだ冬の中．
(yoshi)

愛読者カードご記入のお願い

本誌をお買い上げいただき、誠にありがとうございます。
お手数ですが、今後の誌面づくりの参考にさせていただきたいと存じますので、
各欄にご記入の上、ご返送くださるようお願い申しあげます。

月刊プログラマーズ・ページ READER'S SURVERY CARD　愛読者カード

本誌の号数　　年　　月号

本誌を知った理由
- □1. 書店店頭　□2.他社出版物(　　　)
- □3.小社出版物　□4. DM　□5.口コミ

本誌を購入した書店　　市(区)　　書店

パソコンの所有形態(機種名)
- □1.個人所有　(　　　)
- □2.職場にある(　　　)
- □3.持っていない

よく使用するソフトウェア

よく使用するOS

今月号でおもしろかった記事(いくつでも)

今後取り上げてもらいたい特集のテーマ

貴方が選ぶベスト3

パソコン雑誌	①	②	③
ソフトウェア	①	②	③
パソコン本体	①	②	③

本誌についてのご感想、お気づきの点、その他どのようなことでも結構ですので、ご意見をお聞かせください(「ドラゴン・ポスト＝読者の声」欄に掲載させていただく場合もあります。ペンネームをご希望の方はあわせてご記入ください)。

月刊プログラマーズ・ページ 広告資料請求カード

8	会社(学校)名		9	所属(学部)		10	従業員数	

※下記の該当する番号をご記入ください。

11	業種		12	職種		13	パソコン利用状況	

業種　①製造業　②金融・保険・不動産業　③商社・卸・小売業　④建設業　⑤通信・運輸　⑥システム・ソフト　⑦その他サービス業　⑧公務員　⑨その他

職種　①コンピュータ(ハード)技術者　②コンピュータ(ソフト)技術者　③エレクトロニクス技術者　④プログラマ　⑤その他の技術者　⑥専門職　⑦事務・営業職　⑧経営管理職　⑨商工サービス　⑩教職　⑪学生　⑫その他

パソコン利用状況　①会社で利用　②個人で利用　③会社でも個人でも利用　④利用していない

●希望される広告資料　　年　　月号

広告資料請求No. / 希望項目												
A 購入希望	A	A	A	A	A	A	A	A	A	A	A	A
B 説明希望	B	B	B	B	B	B	B	B	B	B	B	B
C 資料希望	C	C	C	C	C	C	C	C	C	C	C	C

定期購読のご案内

綴じ込みの振替用紙を使って、本誌の定期購読をお申し込みになれます。
1年間12冊＝11,760円（本体11,417円/税343円）
郵送料サービス

郵便はがき

料金受取人払
麹町局承認
3908

差出有効期間
平成4年11月
30日まで

切手不要

102-00

東京都千代田区平河町2-4-14
平河町KSビル
株式会社 翔泳社
マーケティング部 行

月刊プログラマーズ・ページ READER'S SURVEY CARD 愛読者カード

氏名	性別 □1男 □2女	生年月日 19 年 月 日
住所 〒		電話 ( ) －

職業			ユーザー区分
□1ソフト技術者	□2ハード技術者	□3技術系会社員	□1プロ
□4事務系会社員	□5営業系会社員	□6教師	□2マニア
□7一般公務員	□8経営管理職	□9理系学生	□3一般ユーザー
□10文系学生	□11その他（ ）		

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

郵便はがき

料金受取人払
麹町局承認
3910

差出有効期間
平成4年11月
30日まで

切手不要

102-00

東京都千代田区平河町2-4-14
平河町KSビル
株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

＊当社で記入

1	SE	2	受付No.	

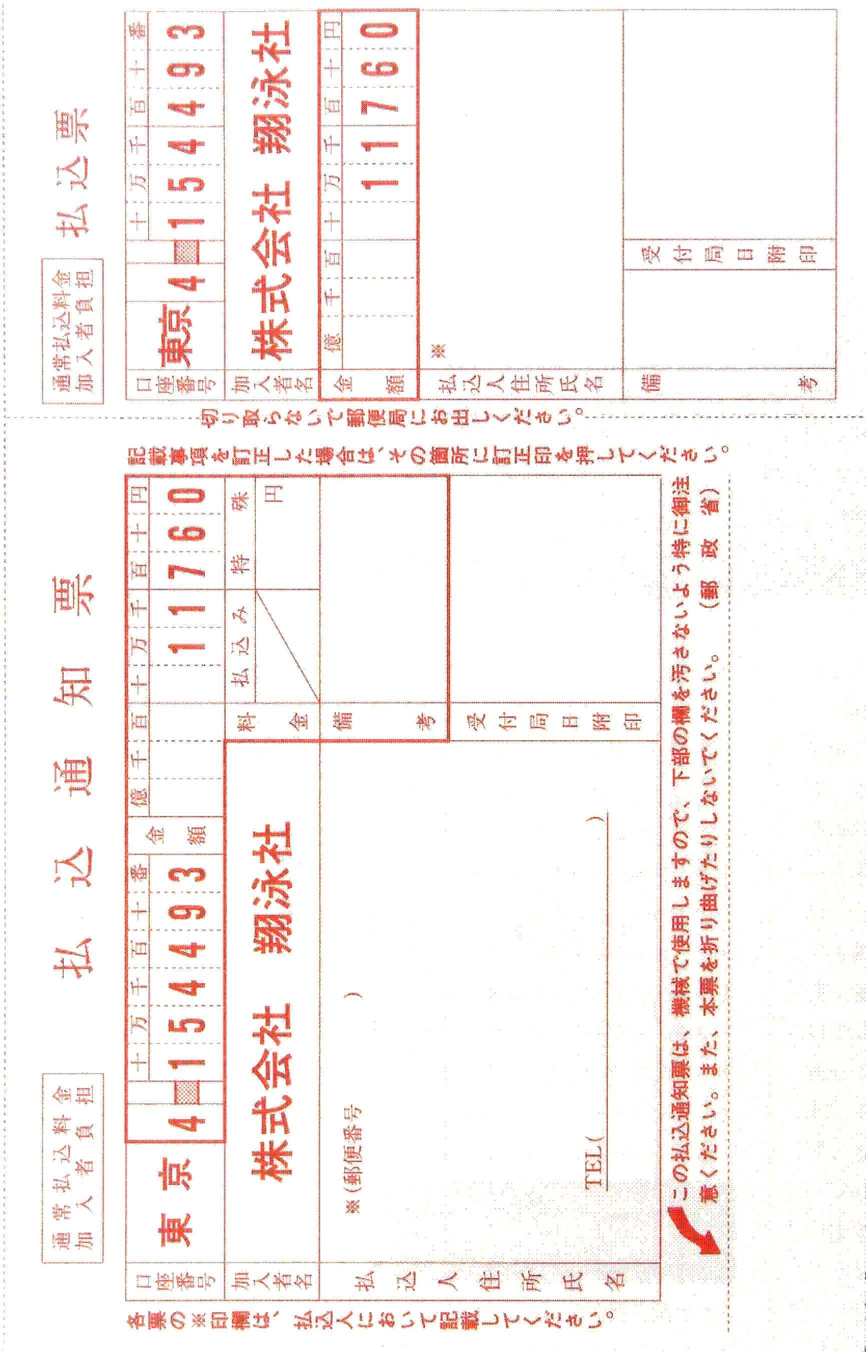
月刊プログラマーズ・ページ広告資料請求カード

3	会員番号	

＊すでに会員番号をお持ちの方はご記入ください。

4	フリガナ 氏名	姓 名	5	生年月日	19 年 月 日

6	自宅住所	〒 電話 ( )	資料送付希望先の番号に○印をつけてください。
7	会社住所	〒 電話 ( )	



ックス

かり馴染みの深い日本語
"かな漢字変換機能"を
はじめて開発、販売した
。以来、同社ではMS−
Xなど、さまざまなOSに
掛け、「日本語処理ならバ
築いてきた。東京・町田の
部長新井俊彦氏を訪ね、開
った。

ね。
に、現在かなり多くのメーカー
きさせていただいているんです
には、"かな漢字変換"を処理す
を自社で持たないメーカーもあ
す。そういうメーカーに対して
までのお付き合い上、何があっ
しいOSが出るときには、"かな
換"を提供しないといけないわ
。これはもう、当社の実績を評
くださるメーカーに対しての責
からね、絶対手を抜けないんで
ちろん、それに付随したパッケ
展開も図っていきたいと考えて
」と今後の抱負を語る新井氏は
続ける。
も、パターン化された同じ仕事
やってるいようではだめだと思うんです。いつも新しいものを意欲的にやっていくという姿勢も大切にしていきたいですね。当社の場合、もともと新しいものに取り組むのが好きな気質もありますから。NeXTもすでに手掛けていて、NeXT用の"VJE Pen"や"VJE γ"、CD−ROM検索の"VJE−CD"を持つ"VJE Pen Super"など、商品化が決定しているものもあります。」

ができるのに、DOSとして使うと日本語はコード入力しかできない。何でいきなりこんなに使いにくくなってしまうのだろう」という疑問に端を発し、何とかDOS上でも日本語入力ができないかと、Ver. 2. 11が出たときに、同社は初めてデバイスドライバを使ったフロントプロセッサを開発するに至った。それが、"VJE−86"である。このVJE−86は、MS−DOS普及期の日

開発をしていきたい」という基本方針を持つ同社。社員の構成も3/4が開発要員というから、その力の入れようがうかがえる。

「今後も、アウトプットに関しては特に大きな変化はないでしょうね。かな漢字変換やワープロ、辞書の検索といった日本語まわりを中心にやっていきたいと思っています。当社が時間をかけてここまで育ててきた得意分野で

定期購読のご案内

■定期購読は、1年間12冊と限定させていただきます。

■購読料のお支払は、この振替用紙をご利用ください。銀行振込、現金書留によるご送金は、ご遠慮ください。

■振替用紙に、ご住所、お名前をご記入のうえ、郵便局の貯金窓口にてお支払ください。

■小社より領収書の発行はいたしませんので、郵便局にて「郵便振替払込金受領証」をお受取ください。

■お申し込みいただいた方には、弊社より直接発送いたしますので、到着が発売日より遅れることがございますが、ご了承ください。

月刊プログラマーズ・ページ　定期購読直接申し込み書

この欄は、加入者あての通信にお使いください。

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

年　　月号より

年間購読料 11,760円（税・送料込み）

■ご希望の購読開始月号をご記入ください。

■送付先（住所・氏名は正確にご記入ください）

住所	□□□-□□	フリガナ	
		氏名	
		TEL	

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。　（郵　政　省）

郵便はがき

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

2	受付No.	

マーズ・ページ広告資料請求カード

＊すでに会員番号をお持ちの方はご記入ください。

名	5	生年月日	19　年　月　日

□10文系学生　□11その他（　　　　）　ユーザー

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

6	自宅住所	〒　電話（　　）	資料送付希望先の番号に○印をつけてください。
7	会社住所	〒　電話（　　）	

綴じ込みの 1

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

広告企画

ちょっと気になるこの会社

株式会社バックス

パソコンユーザーには、もうすっかり馴染みの深い日本語入力フロントプロセッサ。この“かな漢字変換機能”をMS－DOS環境のツールとして、はじめて開発、販売したのが株式会社バックスである。以来、同社ではMS－DOS、OS/2、Windows、UNIXなど、さまざまなOSに対応した製品の開発を次々と手掛け、「日本語処理ならバックス」という確固たる地位を築いてきた。東京・町田の本社に、取締役・営業企画担当部長新井俊彦氏を訪ね、開発・人材に関するお話をうかがった。

MS－DOSと歩んできた10年

パソコンの日本語環境のパイオニア“バックス”が、実はシステムハウスとしてスタートを切ったということをご存知の方はどのくらいおられるだろうか。1981年の設立当初、同社は、何と86系のCPUにCP/Mをのせたハードウェアを作成し、出荷していたのである。ところが、たまたま松下通信工業の受託を受け、JB－3000へMS－DOSの移植を行ったのをきっかけに、さまざまなDOSの移植を手掛けることになり、同社の路線は大きく変化することになった。

その後、「MS－DOSは、日本語ワープロとして使うときは“かな漢字変換”ができるのに、DOSとして使うと日本語はコード入力しかできない。何でいきなりこんなに使いにくくなってしまうのだろう」という疑問に端を発し、何とかDOS上でも日本語入力ができないかと、Ver. 2. 11が出たときに、同社は初めてデバイスドライバを使ったフロントプロセッサを開発するに至った。それが、“VJE－86”である。このVJE－86は、MS－DOS普及期の日本語入力ツールとして重要な役割を果たしながら、次々とバージョンアップされ、今日のVJEに引き継がれている。

そして、当初は受託開発100%としていた同社も、次第に蓄積されたノウハウを活かした自社製品の開発へと移行し、現在はこのVJEシリーズを柱に、各メーカーに対するライセンス販売を主体に事業を展開。また、日本語ワードプロセッサ“VJE Pen”やリアルタイムマルチタスクモニタ“VMX86”などのパッケージソフトの販売にも力を注いでいる。

日本語環境の充実を中心に

「営業力というよりも研究開発力に重きをおいて、OS周辺のソフトウェア開発をしていきたい」という基本方針を持つ同社。社員の構成も3/4が開発要員というから、その力の入れようがうかがえる。

「今後も、アウトプットに関しては特に大きな変化はないでしょうね。かな漢字変換やワープロ、辞書の検索といった日本語まわりを中心にやっていきたいと思っています。当社が時間をかけてここまで育ててきた得意分野ですからね。

それに、現在かなり多くのメーカーと取引きさせていただいているんですが、中には、“かな漢字変換”を処理する部門を自社で持たないメーカーもあるんです。そういうメーカーに対しては、今までのお付き合い上、何があっても新しいOSが出るときには、“かな漢字変換”を提供しないといけないわけです。これはもう、当社の実績を評価してくださるメーカーに対しての責任ですからね、絶対手を抜けないんです。もちろん、それに付随したパッケージの展開も図っていきたいと考えています」と今後の抱負を語る新井氏はこうも続ける。

「でも、パターン化された同じ仕事ばかりやってるいようではだめだと思うんです。いつも新しいものを意欲的にやっていくという姿勢も大切にしていきたいですね。当社の場合、もともと新しいものに取り組むのが好きな気質もありますから。NeXTもすでに手掛けていて、NeXT用の“VJE Pen”や“VJE γ”、CD－ROM検索の“VJE－CD”を持つ“VJE Pen Super”など、商品化が決定しているものもあります。」

魅力はやりがいと充実した仕事環境

開発担当者には、NECのPC－9800シリーズかEPSONが1人に1台と各々の開発対象となるOSが動作するマシンが数台。都合1人あたり2～5台のマシンを駆使することになる。また、今話題のNeXTもすでに10台が導入されており、マルチプラットフォームを謳う同社ならではの充実した開発環境を整えている。

「プロダクトにもよりますが、だいたい4人1チームのプロジェクトで仕事を進めていくケースが多いですね。自社開発が基本ですから、派遣は一切ありません。仕事の流れの大枠は上層部で決めることになりますが、あとは、それぞれ担当者の裁量で仕事を進めてもらっています。それだけ力のあるスタッフが揃っていますからね」と少し誇らし気に語る新井氏。同社では自分の仕事は自分で組み立てていくのが基本。だから、決して大企業のように歯車の一部となることはないのである。責任も重いが、それ以上にやりがいあることは言うまでもない。

「先ほども申し上げましたが、新しいDOSが出ることになれば、当社ではそれに合わせて“かな漢字変換”を出さなくてはなりません。そのため、各メーカーからさまざまな最新情報が入ってきます。つまり、いつも最新技術に触れることができて、メーカーが今後どういう動きをするのかがよくわかるわけです。もちろん、それなりの勉強も必要になりますが、技術者としては、本当に面白い環境にあると思いますね。」

また、同社では、社員が気軽に集まってスキーやテニスへと泊まりがけで出かけるという。「人数が少ないので、話はすぐにまとまりますね。社長や私は、家族も連れて参加しますからね」とにこやかに話す新井氏。仕事を離れても、社員全員が和気あいあいと付き合える。これも、大企業にない同社ならではの魅力といえるのではないだろうか。

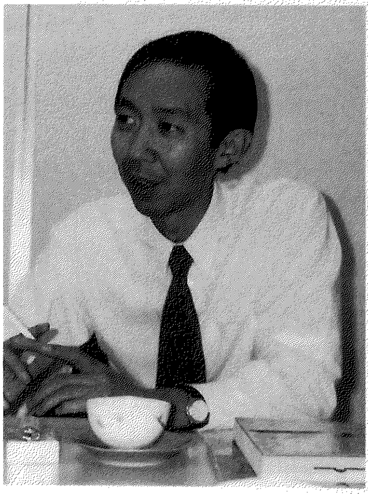
大切なのは目的意識をしっかり持つこと

さらに同社の特色のひとつとして、どこかのメーカーを偏重することなく、各社に対して完全に同じスタンスをとっていることがあげられるだろう。「自分たちでやりたい機械にのせるし、思うとおりにやりたいんです」という新井氏の言葉どおり、同社の場合、大手企業の資本を一切入れておらず、何か新しいことをやろうとした時でも、足枷になるようなことが何もない。つまり、どんなことでもやっていける可能性を秘めているというわけだ。

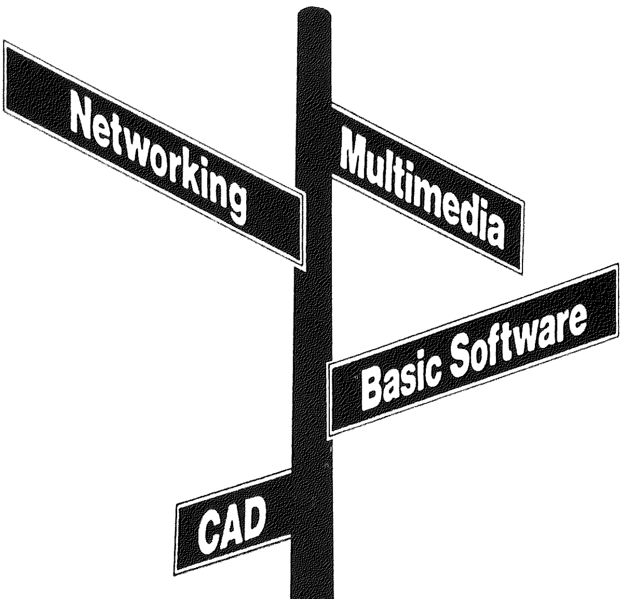
さて、技術力を前面に押し出していきたいという同社だが、その求人像はどんなものなのだろうか。「はっきり言って、新卒者を即戦力とは考えていませんので、特に入社時に技術的なレベルを問うことはありません。そう簡単にコードを書けるようになるわけでもありませんし。一生懸命やる気があって、仕事を真剣にしてくれる人であれば、営業職でも技術職でも入社後にいくらでも覚えてもらえますからね。ただ、何のためにこれをやっているのかということは、いつも考えていて欲しいと思いますね。コーディングしてあるソースコードを見たら、何でここにこう書いてあるんだろうと考えてみなくちゃいけない。もちろん書く時もそうですが、目的意識をはっきり持つことが大切なのではないでしょうか。中途採用の場合でしたら、C言語ぐらいできる人がいいですね。できれば、ひとつのプロジェクトを任せられるような人材が欲しいと思っています。」

入社後は、特に研修のプログラムがあるわけではなく、マンツーマンで個人のレベルに合わせながら、プログラムの仕方を指導。仕事を最初から入れることはせず、大企業にはないきめ細かな教育プロセスで、大切に人を育てているという。

設立から10年、コツコツと積み上げてきた成果が今大きく実を結び、やる気次第で何でもできる懐の深さを感じさせる企業へと成長した同社。さらなる飛躍を期待したい。



取締役・営業企画担当部長　新井俊彦氏



高水準の技術開発にはパーフェクト精神が根幹です。

ワンランク上の提案型ソフトウェアハウス＝エルテックスは、ユーザーや上司の求めている以上の開発を追求していく、我がままプロ集団の組織。常に高レベルの開発を行っていくためには、たえず外部の開発にアンテナを張りめぐらしておかなくてはなりません。その一環として、毎年数名、ラスベガスのコムデックスショー等、1週間程度の視察旅行に参加します。パーフェクトなものを求め続けていく技術者たちに、私たちは、たえず評価を与えていきます。

技術者随時募集

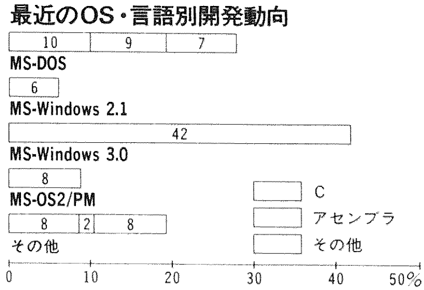
会社概要　ELTEX

設　立	昭和60年12月14日
資本金	1,000万円
事業内容	パソコンを中心としたシステムの提案・開発・販売　①CD-ROM・LD など新しいメディアに対応したシステムの開発 ②ネットワークの端末システムの開発 ③特定業種向けCADの開発・販売 ④基本ソフトの開発
特　徴	大手クライアント向けのシステム開発が多く、ネットワーク関連システムやマルチメディアなど幅広く対応している。また、ＣＡＤを中心とした自社プロダクツにも積極的で、その比率も増加している。基本ソフトの分野では、ハードウェア専門のベンチャー企業とジョイントし、ＯＥＭ製品のＢＩＯＳ・ユーティリティなどを企画から手掛けている。
取引先	三菱電機、三菱自動車、日本航空　他
関連会社	株式会社ソーテック、学校法人福井情報ビジネス総合学院

募集要項　ELTEX

職　種	①システムエンジニア　②プログラマー
資　格	専門卒以上　20〜36歳の方　※Ｃ言語・アセンブラ言語経験者大歓迎　※未経験者でも好奇心・向学心の旺盛な方は歓迎します。
給　与	17万円（専門卒初任給）〜46万円（36歳課長、諸手当含む） 24歳例：23万5000円（情報処理１種手当・住宅手当含む） 30歳例：31万6000円（中級SE、情報処理１種・住宅手当含む）
手　当	時間外手当、役職手当（5000円〜６万円）、住宅手当（１万5000円）、資格手当（２種：5000〜特種：２万円）、通勤手当
賞　与	賞与年２回（平成１年度実績年間５ヵ月）、決算賞与年１回（平成１年度実績１ヵ月）
勤務地	本社（横浜駅近く） ◇派遣・出向はありません。
勤務時間	フレックスタイム制（コアタイム10:00〜16:00）試用期間中は標準時間9:00〜17:45
休日休暇	完全週休２日制（土・日）、祝日、年３回連休（GW・夏期・年末年始）、有給・特別・休暇
福利厚生	各種社会保険完備、退職金制度、財形貯蓄、社員旅行、海外研修、定期健康診断、社員寮、スポーツクラブ法人会員
応募方法	電話連絡の上、履歴書（写真貼付）、職務経歴書を持参し来社ください。郵送も受け付けます。
連絡先	〒220 横浜市西区南幸二丁目20番５号　東伸24ビル５階　℡(045)322-9667　総務課

最近のOS・言語別開発動向



採用HOT・LINE
TEL.(045)322-9667

株式会社エルテックス

ATコマンド一覧

●MD96FS5V(OMRON)

コマンド名	パラメータ	説明
A/	なし	直前に実行したコマンドの再実行
A	なし	アンサーモードで回線接続を行う
B<n>	n=0～2	通信規格,着信時自動速度応答設定(0:CCITT 1:BELL 2:着信時自動速度応答あり)
D<str>	str=電話番号	ダイヤルコマンド
E<n>	n=0,1	コマンド入力時のエコー(0:なし 1:あり)
H<n>	n=0～2	電話回線との接続制御(0:電話回線オンフック 1:TELオンフック,LINEオフフック 2:TEL,LINEオフフック)
I<n>	n=0,1	プロダクトIDコード(0:S29レジスタの内容表示 1:S30レジスタの内容表示)
J<n>	n=0～2	OMRON LINK機能(0:なし 1:あり 2:あり(拡張表示))
K<n>	n=8～11	通信中のキャラクタ長
M<n>	n=0～5	モニタスピーカ(0:常時OFF 1:通信開始までON 2:常時ON 3:常時OFF,着信モニタあり 4:通信開始までON,着信モニタあり 5:常時ON,着信モニタあり)
N		短縮ダイヤルレジスタの設定,表示
O<n>	n=0,1	オンラインモード強制リトレイン(0:オンラインモードへ復帰 1:リトレイン実行後,オンラインモードへ復帰)
P<n>	n=0～2	ダイヤルコマンドをパルス式に設定(0:ダイヤルパルス 1:10pps 2:20pps)
Q<n>	n=0,1	リザルトコード(0:あり 1:なし)
S		Sレジスタの表示,設定
T	なし	ダイヤルコマンドをプッシュ式に設定
V<n>	n=0,1	リザルトコードの表示形式(0:数字形式 1:単語形式)
X<n>	n=0～4	接続完了時のリザルトコード(0:通信速度表示なし 1:通信速度表示あり 2:通信速度表示あり(ダイヤルトーン検出) 3:通信速度表示あり(話中音検出) 4:通信速度表示あり(ダイヤルトーン,話中音検出))
Z<n>	n=0～2	ソフトウェアリセット(0:リセット 1:不揮発メモリの初期化 2:スーパーリセット)
&E<n>	n=0～3	接続完了時の速度表示の選択(0:モデムポート速度表示 1:シリアルポート速度表示(Nタイプ) 2:シリアルポート速度表示(Hタイプ) 3:シリアルポート速度表示(Mタイプ))
&F	なし	工場出荷時の状態に設定
&K<n>	n=0～4	XON/XOFFフロー制御のデータフォーマット(0:自動認識あり 1:自動認識なし 2:7ビット偶数パリティ固定 3:7ビット奇数パリティ固定 4:8ビットパリティなし固定)
&L<n>	n=0,1	ハンドシェーク(0:あり 1:なし)
&M<n>	n=0～6	V.42(0～4:なし 5:V.42リライアブルオンリー 6:V.42オートリライアブル)
&Q<n>	n=0～4	端末速度固定モード時のシリアルポートフロー制御方式(0:なし 1:XON/XOFFフロー制御 2:RS/CSフロー制御 3:CSフロー制御 4:片方向XON/OFFフロー制御)
&T<n>	n=0～6	自己診断機能
&W	なし	設定値を不揮発メモリに記憶
¥A<n>	n=0～3	MNP最大ブロック長(0:64バイト 1:128バイト 2:192バイト 3:256バイト)
¥C<n>	n=0～2	オートリライアブルバッファ機能,フォールバックキャラクタ機能(0:なし 1:オートリライアブルバッファ機能 2:フォールバックキャラクタ機能)
¥D<n>	n=0,1	ディテクションフェーズ(0:あり 1:なし)
¥G<n>	n=0～2	端末速度固定モード時のモデムポートフロー制御(0:なし 1:XON/XOFFフロー制御 2:片方向XON/XOFFフロー制御)
¥J<n>	n=0～2	端末速度固定モード(0:あり 1:なし 2:MNP,V.42接続時のみ端末速度固定モード)
¥K<n>	n=0～5	MNP,V.42接続時のブレーク信号送出によるバッファ動作(0,1:バッファクリア後ブレーク信号送出 2,3:バッファより先にブレーク信号送出 4,5:バッファにブレーク信号を送る)
¥L<n>	n=0,1	トレリスコーデイング(0:なし 1:あり)
¥N<n>	n=0～3	MNP(0,1:なし 2:MNPリライアブルオンリー 3:MNPオートリライアブル)
¥Q<n>	n=0～4	MNP,V.42接続時のシリアルポートフロー制御方式(0:なし 1:XON/XOFFフロー制御 2:RS/CSフロー制御 3:CSフロー制御 4:片方向XON/XOFFフロー制御)
¥R<n>	n=0～2	RI(CI)信号の動作(0:通信開始までON 1:通信終了までON 2:呼出信号に同期)
¥S	なし	ステータス表示
¥V<n>	n=0～2	接続完了時のリザルトコード(0:なし 1:エラー訂正表示あり 2:エラー訂正・データ圧縮表示あり)
%A<n>	n=0～255	オートリライアブルフォールバックキャラクタ(0:なし 1～255:キャラクタ)
%B<str>	str=通信速度	着信時最高通信速度(9600, 4800, 2400, 1200, 300)
%C<n>	n=0,1	MNPのデータ圧縮(0:なし 1:あり)
%E<n>	n=0,1	自動リトレイン(0:なし 1:あり)
%M		Mレジスタの表示,設定
%R	なし	S,Mレジスタ参照
*A<n>	n=0,1	リモートメンテナンス要求設定(0:なし 1:あり)
*C<n>	n=0,1	V.42bisによるデータ圧縮(0:なし 1:あり)
*E<n>	n=0,1	リモートメンテナンス受け付け許可(0:禁止 1:許可)
*L		送出レベル
*P		リモートメンテナンスパスワード
*X	なし	リモートメンテナンス終了

●MD-24FB5V(OMRON)

コマンド名	パラメータ	説明
A/	なし	直前に実行したコマンドの再実行
A	なし	アンサーモードで回線接続を行う
B<n>	n=0～2	通信規格,着信時自動速度応答設定(0:CCITT 1:BELL 2:着信時自動速度応答あり)
D<str>	str=電話番号	ダイヤルコマンド
E<n>	n=0,1	コマンド入力時のエコー(0:なし 1:あり)
H<n>	n=0～2	電話回線との接続制御(0:電話回線オンフック 1:TELオンフック,LINEオフフック 2:TEL,LINEオフフック)
I<n>	n=0,1	プロダクトIDコード(0:S29レジスタの内容表示 1:S30レジスタの内容を表示)
J<n>	n=0～2	OMRON LINK機能(0:なし 1:あり 2:あり(拡張表示))
K<n>	n=8～11	通信中のキャラクタ長
M<n>	n=0～5	モニタスピーカ(0:常時OFF 1:通信開始までON 2:常時ON 3:常時OFF,着信モニタあり 4:通信開始までON,着信モニタあり 5:常時ON,着信モニタあり)
N		短縮ダイヤルレジスタの設定,表示
O<n>	n=0,1	オンラインモード強制リトレイン(0:オンラインモードへ復帰 1:リトレイン実行後,オンラインモードへ復帰)
P<n>	n=0～2	ダイヤルコマンドをパルス式に設定(0:ダイヤルパルス 1:10pps 2:20pps)
Q<n>	n=0,1	リザルトコード(0:あり 1:なし)
S		Sレジスタの表示,設定
T	なし	ダイヤルコマンドをプッシュ式に設定
V<n>	n=0,1	リザルトコードの表示形式(0:数字形式 1:単語形式)
X<n>	n=0～4	接続完了時のリザルトコード(0:通信速度表示なし 1:通信速度表示あり 2:通信速度表示あり(ダイヤルトーン検出) 3:通信速度表示あり(話中音検出) 4:通信速度表示あり(ダイヤルトーン,話中音検出))
Z<n>	n=0～2	ソフトウェアリセット(0:リセット 1:不揮発メモリの初期化 2:スーパーリセット)
&E<n>	n=0～3	接続完了時の速度表示の選択(0:モデムポート速度表示 1:シリアルポート速度表示(Nタイプ) 2:シリアルポート速度表示(Hタイプ) 3:シリアルポート速度表示(Mタイプ))
&F	なし	工場出荷時の状態に設定
&K<n>	n=0～4	XON/XOFFフロー制御のデータフォーマット(0:自動認識あり 1:自動認識なし 2:7ビット偶数パリティ固定 3:7ビット奇数パリティ固定 4:8ビットパリティなし固定)
&L<n>	n=0,1	ハンドシェーク(0:あり 1:なし)
&M<n>	n=0～6	V.42(0～4:なし 5:V.42オートリライアブル 6:V.42リライアブルオンリー)
&T<n>	n=0～6	自己診断機能
&W	なし	設定値を不揮発メモリに記憶
¥A<n>	n=0～3	MNP最大ブロック長(0:64バイト 1:128バイト 2:192バイト 3:256バイト)
¥C<n>	n=0～2	オートリライアブルバッファ機能,フォールバックキャラクタ機能(0:なし 1:オートリライアブルバッファ機能 2:フォールバックキャラクタ機能)
¥D<n>	n=0,1	ディテクションフェーズ(0:あり 1:なし)
¥G<n>	n=0～2	端末速度固定モード時のモデムポートフロー制御(0:なし 1:XON/XOFFフロー制御 2:片方向XON/XOFFフロー制御)
¥J<n>	n=0～2	端末速度固定モード(0:あり 1:なし 2:MNP,V.42接続時のみ端末速度固定モード)
¥K<n>	n=0～5	MNP,V.42接続時のブレーク信号送出によるバッファ動作(0,1:バッファクリア後ブレーク信号送出 2,3:バッファより先にブレーク信号送出 4,5:バッファにブレーク信号を送る)
¥N<n>	n=0～3	MNP(0,1:なし 2:MNPリライアブルオンリー 3:MNPオートリライアブル)
¥Q<n>	n=0～4	MNP,V.42接続時のシリアルポートフロー制御方式(0:なし 1:XON/XOFFフロー制御 2:RS/CSフロー制御 3:CSフロー制御 4:片方向XON/XOFFフロー制御)
¥R<n>	n=0～2	RI(CI)信号の動作(0:通信開始までON 1:通信終了までON 2:呼出信号に同期)
¥S	なし	ステータス表示
¥V<n>	n=0～2	接続完了時のリザルトコード(0:なし 1:エラー訂正表示あり 2:エラー訂正・データ圧縮表示あり)
%A<n>	n=0～255	オートリライアブルフォールバックキャラクタ(0:なし 1～255:キャラクタ)
%C<n>	n=0,1	MNPのデータ圧縮(0:なし 1:あり)
%E<n>	n=0,1	自動リトレイン(0:なし 1:あり)
%M		Mレジスタの表示,設定
%R	なし	S,Mレジスタ参照
*A<n>	n=0,1	リモートメンテナンス要求設定(0:なし 1:あり)
*C<n>	n=0,1	V.42bisによるデータ圧縮(0:なし 1:あり)
*E<n>	n=0,1	リモートメンテナンス受け付け許可(0:禁止 1:許可)
*L		送出レベル
*P		リモートメンテナンスパスワード
*X	なし	リモートメンテナンス終了

●MD-24FP5V(OMRON)

コマンド名	パラメータ	説明
A/	なし	直前に実行したコマンドの再実行
A	なし	アンサーモードで回線接続を行う
B<n>	n=0～2	通信規格,着信時自動速度応答設定(0:CCITT 1:BELL 2:CCITT規格で,V.21まで着信時自動速度応答)
D<str>	str=電話番号	ダイヤルコマンド
E<n>	n=0,1	コマンド入力時のエコー(0:なし 1:あり)
H<n>	n=0～2	電話回線との接続制御(0:電話回線オンフック 1:電話回線オフフック)
L<n>	n=1～3	スピーカ音量(1:小 2:中 3:大)
M<n>	n=0～2	モニタスピーカ(0:常時OFF 1:通信開始までON 2:常時ON)
O<n>	n=0,1	オンラインモード強制リトレイン(0:オンラインモードへ復帰 1:リトレイン実行後,オンラインモードへ復帰)
P	なし	ダイヤルコマンドをパルス式に設定
Q<n>	n=0,1	リザルトコード(0:あり 1:なし)
S		Sレジスタの表示,設定
T	なし	ダイヤルコマンドをプッシュ式に設定
V<n>	n=0,1	リザルトコードの表示形式(0:数字形式 1:単語形式)
X<n>	n=0～4	接続完了時のリザルトコード(0:通信速度表示なし 1:通信速度表示あり 2:通信速度表示あり(ダイヤルトーン検出) 3:通信速度表示あり(話中音検出) 4:通信速度表示あり(ダイヤルトーン,話中音検出))
Y<n>	n=0,1	ロングスペースディスコネクト(0:なし 1:あり)
Z	なし	ソフトウェアリセット
&C<n>	n=0,1	CD信号(0:CD信号を常時ON 1:キャリア検出時のみCD信号をON)
&D<n>	n=0～3	DTR信号に対する動作(0:DTR信号無視(常時ONとして動作) 1:ON→OFFのタイミングで,オンラインモードからコマンドモードへ 2:ON→OFFのタイミングで回線切断 3:ON→OFFのタイミングでリセット)
&E<n>	n=0～1	接続完了時の速度表示の選択(0:モデムポート速度表示 1:シリアルポート速度表示)
&F	なし	工場出荷時の状態に設定
&G<n>	n=0～2	ガードトーン(0:ガードトーン送出なし 1:550Hzのガードトーン送出 2:1800Hzのガードトーン送出)
&L<n>	n=0,1	ハンドシェーク(0:あり 1:なし)
&M<n>	n=0～6	V.42(0～4:なし 5:V.42オートリライアブル 6:V.42リライアブルオンリー))
&P<n>	n=0～2	パルスダイヤル時のパルス速度設定(0,1:10pps 2:20pps)
&S<n>	n=0,1	DR信号(0:DR信号常時ON 1:CCITT勧告に準拠)
&T<n>	n=0～8	自己診断機能
&W	なし	設定値を不揮発メモリに記憶
&Z		短縮ダイヤルの登録
¥C<n>	n=0～2	オートリライアブルバッファ機能,フォールバックキャラクタ機能(0:なし 1:オートリライアブルバッファ機能 2:フォールバックキャラクタ機能)
¥G<n>	n=0～2	端末速度固定モード時のモデムポートフロー制御(0:なし 1:XON/XOFFフロー制御 2:片方向XON/XOFFフロー制御)

¥J<n>	n=0〜2	端末速度固定モード(0:あり 1:なし 2:MNP, V.42接続時のみ端末速度固定モード)
¥K<n>	n=0〜5	MNP, V.42接続時のブレーク信号送出によるバッファ動作(0,1:バッファクリア後, ブレーク信号送出 2,3:バッファより先にブレーク信号送出 4,5:バッファにブレーク信号を送る)
¥N<n>	n=0〜3	MNP(0,1:なし 2:MNPリライアブルオンリー 3:MNPオートリライアブル)
¥Q<n>	n=0〜3	MNP, V.42接続時のシリアルポートフロー制御方式(0:なし 1:XON/XOFFフロー制御 2:RS/CSフロー制御 3:片方向XON/XOFFフロー制御)
¥S	なし	ステータス表示
¥V<n>	n=0〜2	接続完了時のリザルトコード(0:なし 1:エラー訂正表示あり 2:エラー訂正・データ圧縮表示あり)
%A<n>	n=0〜255	オートリライアブルフォールバックキャラクタ(0:なし 1〜255:キャラクタ)
%C<n>	n=0,1	MNPのデータ圧縮(0:なし 1:あり)
%E<n>	n=0,1	自動リトレイン(0:なし 1:あり)
＊C<n>	n=0,1	V.42bisによるデータ圧縮(0:なし 1:あり)

●PV-A96UB5(AIWA)

コマンド名	パラメータ	説明
A/	なし	直前に実行したコマンドの再実行
A	なし	アンサーモードで回線接続を行う
B<n>	n=0,1,15,16,41,42,60,61	呼び出し時の通信規格設定(0:V.22/1200bps 1:BELL 212A/1200bps 15:V.21/300bps 16:BELL 103/300bps 41:V.32/4800bps 42: V.32HDX/4800bps 60:V.32/9600bps 61:V.32HDX/9600bps)
D<str>	str=電話番号	ダイヤルコマンド
E<n>	n=0,1	コマンド入力時のエコー(0:なし 1:あり)
H<n>	n=0,1	電話回線との接続制御(0:オンフック 1:オフフック)
I<n>	n=0〜2	IDコードの表示, ROMチェック(0:IDコード 1:ROMチェック 2:ROMチェック後結果を表示)
L<n>	n=0〜3	スピーカ音量(1:小 2:中 3:大)
M<n>	n=0〜3	モニタスピーカ(0:常時OFF 1:通信開始までON 2:常時ON 3:ダイヤル以外通信開始までON)
N<n>	n=0,1,3,5	接続速度・接続プロトコルフォールバックの設定
O<n>	n=0,1	データモード強制リトレイン(0:データモードに復帰 1:リトレイン実行後, データモードに復帰)
P	なし	ダイヤルコマンドをパルス式に設定
Q<n>	n=0〜2	リザルトコード(0:あり 1:なし 2:オリジネートモードのみあり)
R	なし	強制的にアンサーモードにする
S		Sレジスタの表示, 設定
T	なし	ダイヤルコマンドをプッシュ式に設定
V<n>	n=0,1	リザルトコードの表示形式(0:数字形式 1:単語形式)
W<n>	n=0〜2	ネゴシエーションプロクレスコードの設定
X<n>	n=0〜4	接続完了時のリザルトコード(0:通信速度表示なし 1:通信速度表示あり 2:通信速度表示あり(ダイヤルトーン検出) 3:通信速度表示あり(話中音検出) 4:通信速度表示あり(ダイヤルトーン, 話中音検出))
Y<n>	n=0,1	ロングスペースディスコネクト(0:なし 1:あり)
Z<n>	n=0,1	ソフトウェアリセット(0:プロファイル0でリセット 1:プロファイル1でリセット)
&C<n>	n=0,1	CD(DCD)信号(0:CD信号を常時ON 1:キャリア検出時のみCD信号をON)
&D<n>	n=0〜3	ER(DTR)信号に対する動作(0:DTR信号無視(常時ONとして動作) 1:ON→OFFのタイミングで, オンラインモードからコマンドモードへ 2:ON→OFFのタイミングで回線切断 3:ON→OFFのタイミングでリセット)
&F	なし	工場出荷時の状態に設定
&G<n>	n=0,2	ガードトーン(0:ガードトーン送出なし 2:1800Hzのガードトーン送出)
&K<n>	n=0,3,4	シリアルポートのフロー制御(0:なし 3:RTS/CTSフロー制御 4:XON/XOFFフロー制御)
&L<n>	n=0,1	交換機の設定(0:交換機回線 1:自営回線)
&M<n>	n=0〜3	同期モード・接続プロトコル(0:非同期モード 1,2,3:同期通信モード)
&P<n>	n=1〜2	パルスダイヤル時のパルス速度(1:10pps 2:20pps)
&Q<n>	n=0〜6	接続モードの設定
&R<n>	n=0,1	CS(CTS)信号(0:CTS信号はRTS信号による 1:RTS信号, CTS信号は無視し常にON)
&S<n>	n=0〜2	DR(DSR)信号(0:DSR信号常時ON 1:DSR信号はRS-232Cに従う 2:CD信号のタイミングと等しい)
&T<n>	n=0〜8	自己診断機能
&U<n>	n=0,1	トレリスコーディング(0:あり 1:なし)
&V	なし	不揮発メモリの内容表示
&W<n>	n=0,1	設定値を不揮発メモリに記憶(0:プロファイル0に記憶 1:プロファイル1に記憶)
&X<n>	n=0〜2	送信クロック(0:モデムによる 1:端末による 2:受信キャリアによる)
&Y<n>	n=0,1	電源投入時の読み込む不揮発メモリの指定(0:プロファイル0 1:プロファイル1)
&Z<n>=<str>		短縮ダイヤルの登録

●PV-A24B5(AIWA)

コマンド名	パラメータ	説明
A/	なし	直前に実行したコマンドの再実行
A	なし	アンサーモードで回線接続を行う
B<n>	n=0〜2	通信規格, 着信時自動速度応答設定(0:CCITT 1:BELL 2:CCITTで速度自動選択しない)
D<str>	str=電話番号	ダイヤルコマンド
E<n>	n=0,1	コマンド入力時のエコー(0:なし 1:あり)
H<n>	n=0,1	電話回線との接続制御(0:オンフック 1:オフフック)
I<n>	n=0〜2	IDコードの表示, ROMチェック(0:IDコード 1:ROMチェック 2:ROMチェック後結果を表示)
M<n>	n=0〜3	モニタスピーカ(0:常時OFF 1:通信開始までON 2:常時ON 3:ダイヤル以外通信開始までON)
O<n>	n=0,1	データモード強制リトレイン(0:データモードに復帰 1:リトレイン実行後, データモードに復帰)
P	なし	ダイヤルコマンドをパルス式に設定
Q<n>	n=0,1	リザルトコード(0:あり 1:なし)
S		Sレジスタの表示, 設定
T	なし	ダイヤルコマンドをプッシュ式に設定
V<n>	n=0,1	リザルトコードの表示形式(0:数字形式 1:単語形式)
X<n>	n=0〜4	接続完了時のリザルトコード(0:通信速度表示なし 1:通信速度表示あり 2:通信速度表示あり(ダイヤルトーン検出) 3:通信速度表示あり(話中音検出) 4:通信速度表示あり(ダイヤルトーン, 話中音検出))
Y<n>	n=0,1	ブレーク信号受信時の回線接続状態(0:回線を切断しない 1:回線を切断する)
Z	なし	ソフトウェアリセット
&C<n>	n=0,1	CD(DCD)信号(0:CD信号を常時ON 1:キャリア検出時のみCD信号をON)
&D<n>	n=0〜3	ER(DTR)信号に対する動作(0:DTR信号無視(常時ONとして動作) 1:ON→OFFのタイミングで, オンラインモードからコマンドモードへ 2:ON→OFFのタイミングで回線切断 3:ON→OFFのタイミングでリセット)
&F	なし	工場出荷時の状態に設定
&G<n>	n=0〜2	ガードトーン(0:ガードトーン送出なし 1:550Hzのガードトーン送出 2:1800Hzのガードトーン送出)
&L<n>	n=0,1	交換機の設定(0:交換機回線 1:自営回線)
&M<n>	n=0〜7	MNP, V.42の選択(0:非同期モード 1,2,3:同期通信モード 4:MNPオートリライアブル 5:V.42オートリライアブル 6:MNPリライアブルオンリー 7:V.42リライアブルオンリー)
&R<n>	n=0,1	CS(CTS)信号(0:CTS信号はRTS信号による 1:RTS信号, CTS信号は無視し常にON)
&S<n>	n=0〜2	DR(DSR)信号(0:DSR信号常時ON 1:DSR信号はRS-232Cに従う 2:CD信号のタイミングと等しい)
&T<n>	n=0〜9	自己診断機能
&W	なし	設定値を不揮発メモリに記憶
&X<n>	n=0〜2	送信クロック(0:モデムによる 1:端末による 2:受信キャリアによる)
&Z	なし	短縮ダイヤルの登録
¥A<n>	n=0〜3	MNP最大ブロック長(0:64バイト 1:128バイト 2:192バイト 3:256バイト)
¥B	なし	ブレーク信号の送出
¥G<n>	n=0,1	モデムポートフロー制御(0:なし 1:XON/XOFFフロー制御)
¥J<n>	n=0,1	端末速度固定モード(0:あり 1:なし)
¥K<n>	n=0〜6	ブレーク信号の制御
¥N<n>	n=0〜3	MNP, V.42の選択(0:&M0と同等 2:&M6と同等 3:&M4と同等)
¥O	n=0,1	MNP, V.42の選択(0:MNP 1:V.42)
¥Q<n>	n=0〜3	フロー制御(0:なし 1:XONN/XOFFフロー制御 2:CTSフロー制御 3:RTS/CTSフロー制御)
¥S<n>	n=0,1	ステータス表示(0:ステータス表示1 1:ステータス表示2)
¥T<n>	n=0〜10	無通信監視タイマ(0:なし 1〜10:分単位でタイマを設定)
¥U	なし	プロトコル接続要求を受け入れる
¥V<n>	n=0〜3	接続完了時のリザルトコード(0:なし 1:エラー訂正表示あり 2:エラー訂正・データ圧縮表示あり 3:2に加えシリアルポートのスピード表示)
¥X<n>	n=0,1	透過タイプフロー制御制御(0:なし 1:あり)
¥Z	なし	ノーマルモードに切り替える
%C<n>	n=0,1	MNPのデータ圧縮(0:なし 1:あり)
%R	なし	すべてのレジスタ内容の表示

●COMSTARZ CLUB 24/5(NEC)

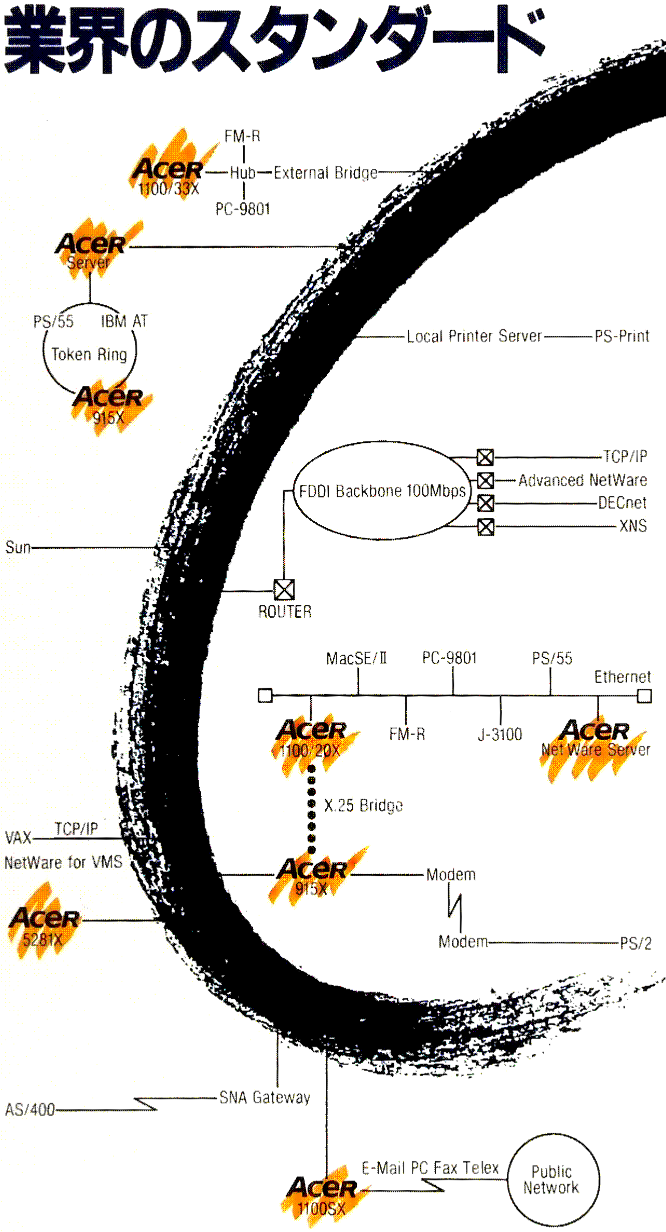
コマンド名	パラメータ	説明
A/	なし	直前に実行したコマンドの再実行
A	なし	アンサーモードで回線接続を行う
D<str>	str=電話番号	ダイヤルコマンド
E<n>	n=0,1	コマンド入力時のエコー(0:なし 1:あり)
H<n>	n=0〜2	電話回線との接続制御(0:電話回線オンフック 1:TELオンフック, LINEオフフック 2:TEL, LINEオフフック)
I<n>	n=0,1	装置コードとROMバージョンの表示
M<n>	n=0〜3	モニタスピーカ(0:常時OFF 1:通信開始までON 2:常時ON 3:常時OFF)
O	なし	オンラインモード強制リトレイン
P<n>	n=0〜2	ダイヤルコマンドをパルス式に設定(0:ダイヤルパルス 1:10pps 2:20pps)
Q<n>	n=0,1	リザルトコード(0:あり 1:なし)
R	なし	強制的にアンサーモードにする
S		Sレジスタの表示, 設定
T	なし	ダイヤルコマンドをプッシュ式に設定
V<n>	n=0,1	リザルトコードの表示形式(0:数字形式 1:単語形式)
X<n>	n=0〜4	接続完了時のリザルトコード(0:通信速度表示なし 1:通信速度表示あり 2:通信速度表示あり(ダイヤルトーン検出) 3:通信速度表示あり(話中音検出) 4:通信速度表示あり(ダイヤルトーン, 話中音検出))
Y<n>	n=0,1	ロングスペースディスコネクト(0:なし 1:あり)
Z	なし	ソフトウェアリセット
&C<n>	n=0,1	CD(DCD)信号(0:CD信号を常時ON 1:キャリア検出時のみCD信号をON)
&D<n>	n=0〜3	ER(DTR)信号に対する動作(0:DTR信号無視(常時ONとして動作) 1:ON→OFFのタイミングで, オンラインモードからコマンドモードへ 2:ON→OFFのタイミングで回線切断 3:ON→OFFのタイミングでリセット)
&E<n>	n=0,1	スペシャルCONNECTメッセージモード
&F	なし	工場出荷時の状態に設定
&G<n>	n=0〜2	ガードトーン(0:ガードトーン送出なし 1:550Hzのガードトーン送出 2:1800Hzのガードトーン送出)
&S<n>	n=0,1	DR信号
&W	なし	設定値を不揮発メモリに記憶
&!<n>	n=0〜2	Sレジスタの表示
&?	なし	コマンドパラメータの表示
¥A<n>	n=0〜3	MNP最大ブロック長(0:64バイト 1:128バイト 2:192バイト 3:256バイト)
¥B<n>	n=1〜255	ブレーク信号長を設定
¥K<n>	n=0〜6	ブレーク信号の制御
¥N<n>	n=0〜6	バッファ動作・エラーフリー動作の設定
¥Q<n>	n=0〜4	フロー制御
¥R<n>	n=0,1	CI信号
¥T<n>	n=0〜255	無通信監視タイマ(0:なし 1〜255:分単位でタイマを設定)
¥V<n>	n=0〜2	接続完了時のリザルトコード(0:なし 1:エラー訂正表示あり 2:エラー訂正・データ圧縮表示あり)
%C<n>	n=0,1	MNPのデータ圧縮(0:なし 1:あり)
%P<n>	n=0,1	PBトーン送出レベル
%T<n>	n=0,1	アンサートーン
%X<n>	n=0〜3	回線通信速度

NetWare

Acer
The Word for Value

パソコンLANにおける業界のスタンダード

THE ARCHITECTURE EXTENDED—AX



世界のパソコンLAN市場をリード

米国Novell社の"NetWare"は、世界のパソコンLAN市場において60%以上の圧倒的なシェアをもつLAN OSです。

このNetWareに日本語が付加されたことによってACERなどのIBM-PC(AT互換機)はもちろんのことAXマシン、PC-9801、FM-R、PS-55、J-3100などの国産PCへの対応も広がっており、日本語・英語両環境でのネットワーク構築を可能といたしました。

NetWareの特徴

NetWareは、ミニコンやメインフレームでしか提供できなかったさまざまな機能をパソコンベースで実現することができます。他のLANシステムと比較した場合の大きな特徴の一つにシステム・フォルト・トレラント(SFT)機能があげられます。例えばファイルサーバーに2台のハードディスクを装着させることによって"ディスクの二重化"が可能であり、常に同じ情報が2台のハードディスクに格納されるため、万一メインのハードディスクに故障が生じても、サブ側のハードディスクが動作し続け通常業務に影響を与えることはありません。(ディスク・デュープレックス機能)

また、書き込みのたびに、そのセクターを読み出し、セクターに不良があるとわかったら、以後そこを使わないようにする不良セクター対策(Hot Fix機能)やOSにとって重要なディレクトリー・テーブルとファイル・アロケーション・テーブルについては、それぞれ別の場所に書き込んでおき、読み出しの際には、コピーと比較してエラー検査をします。(ディレクトリー構造の二重化、ファイル・アロケーション・テーブルの二重化など…)

以上のようにセキュリティと障害対策が他社製品に比べて非常に強力であることがNetWareの特徴といえます。

●NetWareとTCP/IPの共存が可能

TCP/IPプロトコル、PC/NFS機能によりUNIXマシンおよびSUNなどのワークステーションをサーバーにするシステムをサポート。これによってNetWareとTCP/IPソフトが同時に共存可能になりました。

●NetWare製品ラインアップ

「NetWare V.2.2英語版」「NetWare V.2.2日本語版」
「NetWare 386 V.3.1J」「NetWare 386 V.3.11英語版」
以上のNetWareシリーズを取扱っております。なお「NetWare 386 V3.1J」に関しましては、接続可能なPCが他のNetWareと異なりますのであらかじめご確認ください。

世界の舞台で磨かれた

Acerファイル・サーバー Acer1100/33 (PC/AT互換機)をベース



- i80386(33MHz)CPU,RAM 4MB
- 5.25インチ(1.2MB)FDD×1基、
- 100MB～680MB HDD最大2基 実装可能
- 14インチモノクロモニター、モニターI/Fカード
- 16ビットバス用 Ethrnetカード1枚 標準装備
- フロアスタンド他

丸文株式会社

東京都中央区日本橋大伝馬町8-1 丸文ダイヤビル 〒10 第2営業本部 営業第3部 PC&ネットワークグループ TEL (03)3639-9897(ダイヤルイン)

〈資料請求番号P03〉

BORLAND
The Leader in Object-Oriented Programming



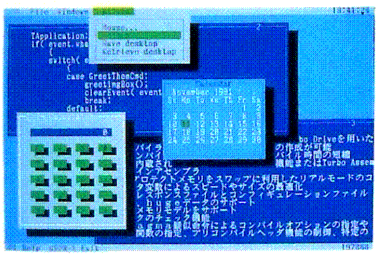
新生ボーランド キックオフ・キャンペーン
BORLAND KICK OFF CAMPAIGN

いま、店頭でBORLAND C++ 2.0を買うと――

ボーランド純正画面制御ライブラリ アプリケーションフレームワークをプレゼント!!

ただ今、店頭にてBORLAND C++ 2.0をお買い求めになり、付属の「アプリケーションフレームワーク無料送付シール」を登録書に貼って返送していただきますと、アメリカでは$199.95(¥28,000)にて提供されております、ボーランド純正画面制御ライブラリ・アプリケーションフレームワーク(ソースコード付属※注)の98版を無料にて送付させていただきます。高速なコンパイルスピードと大規模なアプリケーション開発に適したプロテクトモードのコンパイラを備え、DOSとWindowsのアプリケーションをCとC++で開発できる、プロの開発者のためのBorland C++。この強力な開発環境に、オブジェクト指向テクノロジーを用い、簡単に画面制御を行えるクラスライブラリ・アプリケーションフレームワークを組み合わせれば、開発環境は飛躍的に向上し、プログラマの作業をより楽にすることができます。

APPLICATION FRAMEWORKS



▲Turbo Visionを用いたサンプル画面

プログラマは、自身の創造性やプログラミング能力をアプリケーションの機能や性能の向上のためだけに集中することができるようになります。もう、細かなユーザインターフェース等のプログラミングに多くの時間をさく必要はありません。Windows 3.0用のObject Windows、DOS用のTurbo Vision。必要なプラットフォームに合わせて2種類のフレームワークから選んでいただけます。どちらもマウス対応のオーバーラップマルチウィンドウやメニュー、スクロールバー、ダイアログ、エディタなど、洗練された機能を提供します。アプリケーションフレームワークを使えば、直感的な操作を可能にし、また構造的に優れたユーザフレンドリィなインターフェースを実現できるのです。

■Object Windows：Windows用の画面制御ライブラリ(ソースコード付属※注)

Object Windowsは、グラフィカル・ユーザインターフェースを実現するために、複雑なAPIと格闘しなければならないという面倒な作業を取り除き、Windows 3.0のアプリケーション開発を合理化します。ボーランド独自の、Windowsメッセージを処理するダイナミックディスパッチの機能により、Windowsプログラミングのモデルを簡素化しているのです。Object Windowsにより、MDI(マルチドキュメントインターフェース)やカスタムコントロールへのアクセスなどのWindowsの特徴的な機能を使用したプログラムを簡単に作成できます。新しくWindowsのプログラミングを始められる方でも、古い方法の経験がある方でも、Object Windowsがプログラミングデザインに提供する新次元の自由度の高さや柔軟性を堪能していただけるでしょう。

■Turbo Vision：DOS用の画面制御ライブラリ(ソースコード付属※注)

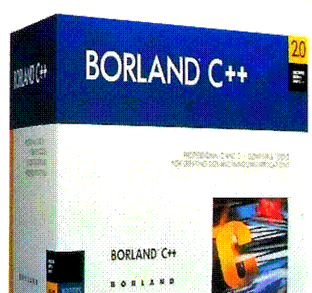
Turbo Visionは、WindowsにおけるObject Windowsと同じように、構造的に優れたユーザインターフェースを、キャラクターベースのDOSアプリケーションに提供します。ほんの少しのコードで、機能的なマウス対応の、位置・サイズ可変なオーバーラップマルチウィンドウやプルダウンメニュー、ダイアログボックス、ボタン等を駆使したイベントドリブン・アプリケーションを、プログラマが望むように実現することができるのです。Turbo Visionを使用すれば、簡単に現代的でエキサイティングなDOSアプリケーションを構築できます。

■価格：BORLAND C++ 2.0アプリケーションフレームワーク・キャンペーンパック…¥98,000(税別)

★価格据え置き!お買い得です!!

※注…ソースコードに関しましては、サポートは一切行われません。ご了承ください。

★さらに、もれなく「プログラミングWINDOWS」(アスキー出版局)をプレゼント!!



BORLAND® C++ 2.0

●製品に関するお問い合せは、総販売元(株)マイクロソフトウェア アソシエイツ TEL.03-3486-1411までお願いします。

ボーランド株式会社
〒107 東京都港区南青山7-8-1 小田急南青山ビル TEL.03-3486-1400(代)
※社名変更いたしました。

T1017777020987
〈資料請求番号P04〉
雑誌コード17777-02